UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

Giovanni Antonio Tomaso Ferreira Rotta

SMART-COMM - PROPOSTA DE UMA ARQUITETURA DE UM SISTEMA DE COMUNICAÇÃO ENTRE DISPOSITIVOS COMPUTACIONAIS HETEROGÊNEOS EM UMA CASA INTELIGENTE

Florianópolis

2017

Giovanni Antonio Tomaso Ferreira Rotta

SMART-COMM - PROPOSTA DE UMA ARQUITETURA DE UM SISTEMA DE COMUNICAÇÃO ENTRE DISPOSITIVOS COMPUTACIONAIS HETEROGÊNEOS EM UMA CASA INTELIGENTE

Trabalho de conclusão de curso submetido ao Curso de Bacharelado em Ciências da Computação para a obtenção do Grau de Bacharel em Ciência da Computação.
Orientador: Prof. Dr. Mario Dantas

Florianópolis

2017

Catalogação na fonte elaborada pela biblioteca da
Universidade Federal de Santa Catarina

A ficha catalográfica é confeccionada pela Biblioteca Central.

Tamanho: 7cm x 12 cm

Fonte: Times New Roman 9,5

Maiores informações em:
http://www.bu.ufsc.br/design/Catalogacao.html

Giovanni Antonio Tomaso Ferreira Rotta

SMART-COMM - PROPOSTA DE UMA ARQUITETURA DE UM SISTEMA DE COMUNICAÇÃO ENTRE DISPOSITIVOS COMPUTACIONAIS HETEROGÊNEOS EM UMA CASA INTELIGENTE

Este Trabalho de conclusão de curso foi julgado aprovado para a obtenção do Título de "Bacharel em Ciência da Computação", e aprovado em sua forma final pela Curso de Bacharelado em Ciências da Computação.

Florianópolis, 8 de junho 2017.

Banca Examinadora:

Prof. Dr. Mario Dantas
Orientador

Prof. Frank Augusto Siqueira

Prof. Roberto Willrich

AGRADECIMENTOS

Agradeço a todos que me acompanharam durante todos estes anos de árdua jornada na graduação.

Aos colegas de disciplinas, que muitas vezes dividiram comigo tempo e angústia, em prol do conhecimento, Ana, Bernardo, Guilherme, João, Nakayama e Wagner.

Também aos Amigos que durante este tempo, dividiram comigo momentos de alegria e também de tristeza, fundamentais para minha formação como pessoa e profissional, Arthur, Fabiano, João, Leandro, Matheus, Maria Eduarda e Rodrigo.

Agradeço aos dois mestres de profissão que tive a oportunidade de trabalhar, e que me acolheram como estagiário e me transmitiram muitos de seus conhecimentos, Ricardo e Eduardo.

Agradeço ao meu orientador Mario e a professora da UFSM Andrea, por acreditarem e fornecerem todo o apoio necessário para o desenvolvimento da minha idéia.

Mas principalmente, aquelas pessoas que sempre acreditaram em mim, respeitaram minha decisões e e que se não fosse por elas eu não teria conseguido chegar até o fim, e nem ao começo. Agradeço a meu pai João, minha mãe Janete, por todo suporte e apoio durante esses anos.

A man who has no imagination has no wings.

Muhammad Ali

RESUMO

Casas e edifícios inteligentes são conceitos já bem maturados dentro da academia e até da sociedade em geral, porém na prática, continuam sendo uma realidade bem distante. Este trabalho foi desenvolvido visando facilitar a concepção de um sistema de casa inteligente além de buscar a integração deste com o conceito de internet das coisas. Foi proposto uma arquitetura para um sistema ubíquo, a fim de solucionar diversos desafios existentes em relação a internet das coisas e casas inteligentes. Foram relacionadas técnicas e arquiteturas computacionais visando a interoperabilização de diferentes protocolos de comunicação entre os dispositivos do sistema. Um protótipo foi desenvolvido para validar a viabilidade da sistema. A proposta oferece uma base em termos computacionais e de comunicação para o gerenciamento de dispositivos e usuários, interoperabilidade, portabilidade de plataforma, context-awareness e segurança.

Palavras-chave: Casas Inteligentes. Internet das Coisas. Computação Ubíqua. Context-Aware. Microservice.

ABSTRACT

Smart homes and bulding are concepts well matured within the academy and even society in general, but in practice, they remain far from reality, aiming to facilitate a design of smart homes and also seeking an integration with the growing concept of the internet of things this paper was created. It proposes a ubiquitous system to solve various nowadays challenges related to the internet of things and smart homes, using an heterogeneous communication to relate techniques and computational architectures with existing and low cost devices. A prototype was designed to validate the proposal, using microservice to treat different types of communication protocol in the IoT devices and a token security approach. Offering a computational and communication base for device management, interoperability, portability Platform, context awareness and security.

Keywords: Smart Home. Internet of things. Ubiquitous Computing. Context-Aware. Microservice.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

6LowPAN - IPv6 over Low power Wireless Personal Area Networks
API - Application Programming Interface
CoAP - Constrained Application Protocol
CPU - Central Processing Unit
EDA - Arquitetura Event-Driven
E/S - Entradas e Saídas
HTTP - Hypertext Transfer Protocol
IEEE - Instituto de Engenheiros Eletricistas e Eletrônicos
IoT - Internet das Coisas
IP - Internet Protocol
JS - Javascript
MQTT - Message Queuing Telemetry Transport
PAN - Personal Area Network
PLC - Programmable Logic Controller
REST - Transferência de Estado Representativo
RFC - Request for Comments
SBC - Single Board Computer
TCP - Transmission Control Protocol
UDP - User Datagram Protocol
URI - Identificador Uniforme de Recursos
URL - Uniform Resource Locator
WPAN - Wireless Personal Area Network

SUMÁRIO

1 INTRODUÇÃO

O conceito de casas inteligentes, ou *smart-homes*, passou por diversas transformações até chegar na idéia atual acerca do tema. Em 1935 uma casa inteligente era aquela onde cada cômodo possuía ponto de eletricidade, em 1955 eram as que possuíam televisão e telefone, no conceito atual, a casa inteligente é aquela que possui um computador por cômodo (WEISER, 1996), este atuando em plano de fundo, fazendo uso dos conceitos de computação ubíqua (WEISER, 1999). Conceito de computação onipresente, na qual sua utilização é realizada em todos os lugares e mesmo quando utilizamos seu serviço, abstrairmos a presença dos computadores.

Apesar de já ser uma realidade em nossa sociedade, as casas inteligentes são escassas e pouco difundidas, é raro conhecer alguém que possua uma, mesmo existindo produtos, soluções no mercado e investimentos de pesquisas pelas grandes representantes da indústria. A principal causa da não popularização das casas inteligentes é o custo dos produtos disponíveis no mercado, principalmente em países subdesenvolvidos, nos quais o custo é muito elevado e nem sempre oferecem uma solução completa, precisando adquirir vários dispositivos diferentes, muitas vezes de diferentes fabricantes, fazendo com que sejam difíceis de integrar e instalar. Por outro lado, existem diversas tecnologias e dispositivos de baixo custo disponíveis que podem exercer as funções de uma casa inteligente se dispostos de maneira adequada, como microcontroladores, SBC(*single board computer*), sensores e atuadores além diversos softwares abertos para integração destes.

O fácil acesso a estas tecnologias e a possibilidade de com elas viabilizar um sistema de casa inteligente, que ofereça uma solução completa e de baixo custo são as principais motivações para este estudo.

1.1 OBJETIVO

O objetivo da pesquisa é elaborar e implementar um sistema de comunicação entre dispositivos de uma casa inteligente, além de oferecer a estes uma base para a realização de tarefas e para aplicações context-aware. O sistema deve ainda possuir baixo custo de implantação e utilizar estratégias para alocação de recursos de nodos ociosos e gerenciamento de dados distribuídos, partindo da premissa que um sistema de casa inteligente é um sistema computacional, no qual, de

forma ubíqua, atende as necessidades do morador da casa, através de controles de automação, sensores e serviços.

O sistema precisa fundamentar uma base capaz de atender alguns requisitos, considerados essenciais para uma casa inteligente. Os requisitos podem ser divididos em sensores/entradas, atuadores/saídas e serviços, dispostos na tabela 1, e serão utilizados para servir os moradores da casa da melhor maneira possível. É necessário que o sistema possa suportar os sensores/entradas, estes são importantes para monitorar e ativar eventos que resultarão em um ação dos atuadores e serviços em uma aplicação context-aware. Os atuadores são responsáveis pela automação da casa, como motores de janelas e portões, e acendimento de lâmpadas. Os serviços, por sua vez, são responsáveis por atender as necessidades dos moradores, fornecendo por exemplo, músicas, vídeos, notícias, previsão do tempo, entre outros.

1.2 OBJETIVOS ESPECÍFICOS

Dado os requisitos da aplicação, podemos então definir alguns dos objetivos específicos da pesquisa, necessários para que a aplicação alcance as especificações necessárias para o objetivo geral.

- Pesquisar e realizar uma pesquisa bibliográfica sobre o assunto de *smart home* e internet da coisas, a fim de se propor uma solução para os problemas comuns a estas tecnologias.

- Propor uma arquitetura de um sistema para *smart home*, implementando um protótipo funcional do sistema e verificando sua viabilidade.

- Estabelecer a conexão entre os dispositivos usados para compor o sistema, realizando trocas de mensagens entre eles e delegando e recebendo tarefas, de modo a utilizar da melhor forma possível o processamento total do sistema.

- Execução de comandos na aplicação, e este comando deve refletir em ações dos dispositivos para realização das tarefas.

Um ponto importante a ser mencionado é que o sistema sugerido não realiza nenhuma tarefa relacionada a inteligência artificial, e por isso, segundo alguns autores, é errado defini-la como uma aplicação ‘smart’. Entretanto, a aplicação pode servir como uma ferramenta

auxiliar, permitindo que outras aplicações de caráter autônomo e inteligente realizem tarefas, gerenciando a troca de mensagens e alocação de recursos e dados destas aplicações.

O estudo pode ser dividido em 2 etapas. A primeira parte trata-se de uma pesquisa na literatura sobre a modelagem e arquitetura que melhor atenda os requisitos do projeto, explicando então as escolhas dos tipos de conexões, topologias da rede, protocolos e softwares.

A segunda etapa consiste na aplicação da arquitetura proposta, implementando um protótipo funcional do sistema e verificando sua viabilidade.

2 SMART HOME

Para que seja possível explicar da melhor maneira o conceito *smart home*, é necessário entender alguns conceitos essenciais, explicados neste capítulo, a fim de se obter uma base sobre o assunto para entender as necessidades do sistema. Começaremos explicando o conceito de computação Ubíqua, idealizada por Mark Weiser, onde a tecnologia está impregnada em nosso modo de agir e pensar, sem nem nos darmos conta de que a estamos utilizando. É preciso explicar também o conceito de Internet das coisas, que será utilizado no sistema para captar dados e realizar tarefas mecânicas.Também serão explicados *context-aware* e *fog computing*, dois conceitos utilizados na proposta, e por fim, o mapeamento das principais funcionalidades de uma casa inteligente.

2.1 COMPUTAÇÃO UBÍQUA

A computação ubíqua é apontada como a terceira onda computacional, sendo que a primeira onda ocorreu entre 1940 até meados de 1980, caracterizada por uma grande quantidade de pessoas utilizando o mesmo computador, chamados mainframes. Esta onda perdurou até o surgimento dos primeiros computadores pessoais. A segunda onda representou o relacionamento pessoa computador, 1 para 1, onde cada indivíduo possuía o seu próprio computador pessoal. A terceira onda, na qual se encontra em fase inicial é caracterizada por uma pessoa sendo servida por diversos computadores, por onde quer que ela esteja (WEISER, 1996).

É isto que se almeja alcançar com a tecnologia estabelecida para uma *smart-home*, fazer com que os usuários a utilize sem se dar conta, integrando ela com o cotidiano, a ponto de se perguntar como seria a vida sem ela. Weiser utilizou uma analogia com a linguagem escrita para explicar a tecnologia ubíqua:

> "Considerando a escrita, talvez a primeira representação de tecnologia da informação: A habilidade de representar uma linguagem falada em símbolos como maneira de guardar a longo prazo uma informação além dos limites da memória de um indivíduo. Hoje esta tecnologia é ubíqua em países industrializados. Não apenas em livros, revistas e jornais convém a escrita de informações, mas também em placas de ruas, anúncios, placas de lojas e até mesmo graffiti, papéis

> de balas são repletos de escritas. Essa presença constante em plano de fundo da 'tecnologia literária' não necessita nenhuma atenção especial, a informação está implícita e pronta para o uso. É difícil imaginar a vida moderna de outro jeito."

Segundo Weiser, iremos abstrair a presença dos computadores em nosso cotidiano, e na maior parte do tempo ignorar sua presença, mesmo quando utilizamos seu serviço. Computadores serão como janelas, pisos, livros, canais de televisão, ligações de telefone, etc. ou seja, parte integrante da casa. Ele não irá substituir o que já existe, mas trará novas maneiras de usar, mais fácil, simples e divertidas (WEISER, 1996).

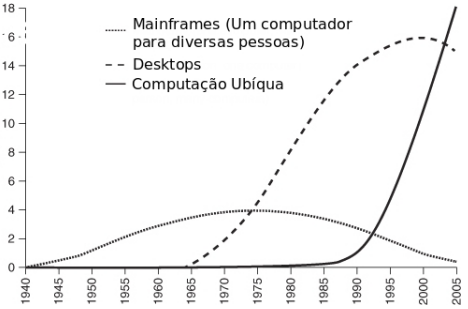

Figura 1 – Tendências computacionais segundo Weiser

Fonte: Fonte Própria

2.2 INTERNET DAS COISAS

Conceito fundamental para entender o comportamento dos objetos, sensores e atuadores em uma casa inteligente, a internet das Coisas, também conhecida como IoT, do inglês *Internet of Things*, é considerada a próxima grande tendência para o mundo da internet (RATHORE et al., 2015). Tal relevância, nos leva a crer que o conceito de casas inteligentes será baseado na intercomunicação entre os diferentes aparelhos eletrônicos, assim como também nos serviços multimídias oferecidos por estes (RATHORE et al., 2015), nas casas inteligentes baseadas em IoT. IoT é um paradigma tecnológico, que tem por finalidade conectar aparelhos eletrônicos utilizados em nosso cotidiano, através da internet.

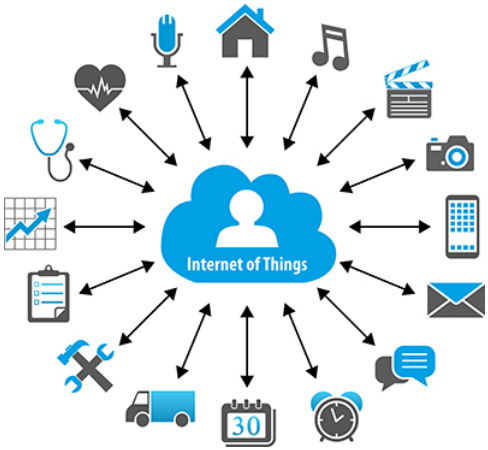

Figura 2 – Representação do universo IoT

Fonte: www.semielectronics.com/tag/sensors

A idéia básica é a presença pervasiva ao nosso entorno de uma variedade de objetos, como sensores, atuadores, *smartphones*, nos quais são capazes de interagir uns com os outros e cooperar com objetos vizinhos a fim de atingir um objetivo em comum (ATZORI, 2010). Ou seja, um ambiente onde a internet estará em todos os lugares e objetos.

A IoT tem o potencial de mudar a maneira como vivemos, do ponto de vista do usuário final ela terá influência tanto no campo doméstico quanto no trabalho.

Um dos objetivos da IoT é rastrear e coletar dados de um determinado objeto, de modo a aumentar a eficiência e facilitar sua utilização no dia-a-dia. Podemos citar como exemplo uma colher, que pesa a quantidade de comida a ser ingerida, podendo repassar ao usuário a quantidade de calorias e nutrientes que ele ingere por dia, e quanto ele vai precisar gastar em energia para manter o peso. Estes objetos podem então compartilhar suas informações com outros objetos ou aplicativos, no caso da colher, ela poderia passar as informações coletadas a bicicleta ergométrica, para que esta já deixe calculado a quantidade de quilômetros que o usuário da colher deverá correr naquele dia.

Este trabalho propõe estabelecer uma rede de conexões entre estes objetos e os outros dispositivos da casa inteligente, a fim de integrá-los. Porém, assim como qualquer tecnologia recente, a IoT também apresenta limitações. Não existe hoje um protocolo universal para os

diferentes tipos de objetos e aplicações, tornando difícil conectar todos os dispositivos em uma mesmo sistema. Este trabalho apresenta um estudo sobre os principais protocolos IoT e sua importância em uma casa inteligente.

2.3 CONTEXT-AWARENESS

A origem do termo é datada de 1994, por Schilit e Theimer e foi definida como “A habilidade que os usuários de aplicações móveis tem em descobrir e reagir a mudanças no ambiente em que ele esteja situado"(BISGAARD; HEISE; STEFFENSEN, 2004).

Desde o nascimento do conceito de computação ubíqua e posteriormente com o surgimento do termo em 1994, *context-awareness* tem se estabelecido como área de pesquisa dentro da ciência da computação. Diversos pesquisadores têm proposto diferentes explicações e conceitos sobre o tema, o mais amplamente aceito é a definição dada por Abowd, em 1999 (PERERA; ZASLAVSKY; CHRISTEN, 2014).

Segundo Abowd, a definição de contexto é qualquer informação que pode ser usada para caracterizar uma situação de uma entidade, sendo a entidade uma pessoa, lugar, ou objeto, considerados relevante para a interação entre o usuário e a aplicação, incluindo o próprio usuário ou a aplicação.

Enquanto para o mesmo autor, a definição de *context-awareness* é quando um sistema utiliza algum contexto para prover informações relevantes e/ou serviços para o usuário, onde a relevância depende da tarefa do usuário.

Este conceito se tornou um recurso importantíssimo da computação ubíqua e pervasiva. A maior concentração de trabalho em relação a este conceito nos últimos anos é em envolver sistemas computacionais, como, aplicações desktop, web, *smartphones* e pervasiva/ubíqua com a Internet das Coisas (PERERA; ZASLAVSKY; CHRISTEN, 2014). *Context-aware computing* nos permite guardar informações do contexto de um sensor, e depois analisar tal informação e entendê-lo. Compreender esse contexto torna mais fácil a execução de uma comunicação entre dispositivos, um elemento crucial na visão IoT (PERERA; ZASLAVSKY; CHRISTEN, 2014).

Quando um grande número de sensores são dispostos e começam a gerar dados, as aplicações tradicionais de captura de contexto tornam-se inviáveis, pelo fato de não conseguirem gerenciar esse grande número. Como solução, pesquisadores estão introduzindo diversos mid-

dlewares, cada um deles focado em diferentes aspectos da IoT, como gerência de *middlewares*, interoperabilidade, *context-awareness*, segurança e privacidade além de muitos outros. Mesmo quando uma única solução engloba diferentes aspectos, um middleware ideal ainda não foi completamente desenvolvido (PERERA; ZASLAVSKY; CHRISTEN, 2014).

Este trabalho visa utilizar o conceito de *context-aware* no sistema de Casas Inteligentes proposto, integrando-o com os outros conceitos explicados até então.

Diferentes fatores determinam como serão as funcionalidades oferecidas pela casa inteligente, sendo o principal fator o indivíduo que receberá o serviço, com suas informações e preferências, porém outras condições são também fundamentais, como por exemplo, a temperatura, horário, local e outras variadas informações sobre o ambiente e o mundo. Grande parte destas informações são coletada através de dados pré cadastrados e armazenados em um banco de dados, porém outras informações são coletadas em tempo real, como por exemplo a temperatura e umidade do ambiente. Essa coleta se dá através de sensores, conectados a um controlador lógico(processador). Estas informações coletadas compõem o contexto de ambiente necessário para uma aplicação *context-aware*.

2.4 FOG COMPUTING

De acordo com a perspectiva da Cisco, *fog computing*(névoa) pode ser considerada uma extensão do paradigma de computação em nuvem, uma rede que se encontra na borda da rede (DEY; ABOWD, 1999). Uma plataforma altamente virtualizada, que provê computação, armazenamento e serviços de rede entre dispositivos e o tradicional servidor *cloud* (DEY; ABOWD, 1999).

Outra definição, ainda debatida por pesquisadores, é a de (VAQUERO; RODERO-MERINO, 2014), "Um cenário onde um enorme número de dispositivos ubíquos e descentralizados se comunicam e cooperam entre eles e com a rede para executar armazenamento e processamento de tarefas sem a intervenção de terceiros. Estas tarefas podem ser suportadas por funções básicas de rede ou serviços e aplicações executadas em ambientes *sand-box*. Usuários que alocam parte de seus dispositivos para hospedar estes serviços recebem incentivos.".

Processamento, armazenamento e recursos de rede são características de ambos: *Cloud* e *Fog computing*, porém existem diversas diferenças que mostram que a névoa não é uma extensão trivial da nu-

vem. Para isso alguns exemplos serão apontados (VAQUERO; RODERO-MERINO, 2014):

- Localização na borda e baixa latência. A *Fog* tem sua origem no suporte a serviços de *endpoints* na borda da rede, incluindo aplicações de baixa latência, jogos e *streaming* de vídeo (VAQUERO; RODERO-MERINO, 2014).

- Distribuição Geográfica. Em contraste com a *Cloud* centralizada, os serviços e aplicações focados na *fog* são em sua vasta maioria distribuídos (VAQUERO; RODERO-MERINO, 2014).

- Redes de sensores de larga escala para monitorar ambientes e Smart Grid são outros exemplos de sistemas distribuídos que precisam de processamento e armazenamento distribuído.

- Interações em tempo real. Aplicações em *Fog* preferem interações em tempo real a processamento *batch*.

- Predominância de acesso sem fio à rede.

Do ponto de vista de um construtor de rede, o maior desafio que aplicações de IoT trazem é a ligação entre os dispositivos e os *data centers* que podem analisar e gerar as informações relevantes para o sistema. Normalmente, os dispositivos IoT falam com um pequeno *gateway* nas proximidades e este pode ter uma conexão tênue e intermitente com a Internet.

A arquitetura do sistema proposto utiliza o conceito de *fog computing*, visando fornecer aos objetos IoT a possibilidade de terem seus dados processados localmente em dispositivos de borda, de forma distribuída, antes de serem enviados a um servidor remoto, permitindo uma resposta mais rápida e menos consumo de processamento na nuvem. Este processamento local configura-se em grande parte a funcionalidades *context-awareness*, onde o tempo de resposta influencia no comportamento da casa. Por exemplo, ao detectar a presença de chuva, as janelas da casa devem ser fechadas.

2.5 FUNCIONALIDADES DE UMA CASA INTELIGENTE

Para melhor exemplificar as funcionalidades de um sistema computacional para casas inteligentes, foram utilizados casos de uso e exemplos, de modo a facilitar o entendimento e compreensão do tema.

Os exemplos utilizados retratam o cotidiano de uma pessoa residente de uma casa equipada com um sistema para casas inteligentes: 8 horas da manhã começa o dia do morador da casa inteligente, e para isso, ela ajusta previamente seu horário de despertar dentro do sistema. O meio de ajuste é variado, podendo ser por um *web browser*, aplicativo de celular, ou por comando de voz, capturado por um microfone localizado no teto do cômodo. A casa inteligente então dispara um alarme, cuja melodia do toque se ajusta de acordo com os compromissos que a pessoa tem pela manhã. Nos finais de semana por exemplo, onde as manhãs são livres de compromissos, o som do alarme poderia ser de passarinhos cantando, porém como é uma segunda feira e esta pessoa tem um compromisso pela manhã, a casa inteligente escolhe uma música agitada para que o habitante possa despertar mais rapidamente.

Ao chegar na cozinha para tomar seu café da manhã, a casa inteligente começa a preparar o café expresso assim que verifica a presença do morador no ambiente. A partir de um comando de voz, a televisão da cozinha se acende e começa a transmitir um serviço de notícias RSS.

No banheiro, através de outro comando, o morador regula a temperatura da água do banho, já a temperatura do piso e do ambiente é regulada automaticamente, de acordo com dados previamente estabelecidos pela pessoa e verificado pelo sensor de temperatura.

Enquanto isso, todas as janelas da casa se abrem de forma automática, de acordo com uma rotina previamente estabelecida para que isso se repita todos os dias da semana. No jardim, de forma análoga, começa a irrigação do gramado.

Ao chegar na garagem, pela captura de um sensor de presença, uma luz se acende automaticamente para auxiliar no manuseio da chave. Já o portão é acionado pelo celular.

Durante o dia, o morador pode verificar o nível de consumo de energia, nível de insolação, umidade, temperatura e outros dados, capturados por sensores espalhados pela casa. A consulta é feita através de um aplicativo de celular ou *webbrowser* direto de seu local de trabalho, do mesmo modo é possível acompanhar as imagens capturadas pelas câmeras instaladas pela casa, e até mesmo, enviar comandos para casa como, por exemplo, a suspensão do trabalho agendado para o aspirador robô.

Os casos de uso exemplificados acima, nos ajudam a compreender melhor o propósito e as funcionalidades de uma casa inteligente. O sistema proposto neste trabalho visa tornar possível a comunicação entre os dispositivos, de modo a permitir e oferecer uma base sólida para sistemas e aplicativos.

Tabela 1 – Tabela de funcionalidades de uma casa inteligente

Sensores/Entradas	**Atuadores/Saídas**	**Serviços**
Presença	Motores elétricos	Notícias
Luminosidade	Relés	Tempo/Clima
Umidade	Acendedor Lareira	Videos/Stream
Temperatura	Irrigação Jardim	Circuito interno
Fumaça	Eletrodomésticos	Músicas
Pressão/Vazão	Aquecedores	Filmes/Séries
Consumo Energia	Saída de Vídeo	Comando Voz
Teclado/senha	Saída de Som	Cenários de iluminação
Interruptor	Alarme	Navegação Internet
Entrada Vídeo/Câmera		
Entrada Microfone		

Através dos casos de uso e da tabela 1, é possível estabelecer pré-requisitos para os dispositivos que compõem o sistema. Assim como a utilização de objetos IoT, representado por sensores e de atuadores, é necessário um *gateway*, equipado com alguns recursos, como, acesso a internet, para captação dos serviços oferecidos(tempo/clima, notícias e etc..) e entrada e saída de áudio e vídeo.

É importante ressaltar que o objetivo da pesquisa não é a implementação do sistema descrito no caso de uso, porém utilizá-lo como base para estabelecer alguns requisitos para a proposta. Como já explicado anteriormente, a pesquisa tem como objetivo a implementação de um sistema para comunicação de forma heterogênea entre os dispositivos de uma casa inteligente, fornecendo uma base para a realização das tarefas explicadas que seja de baixo custo e utilizando estratégias para alocação de recursos de nodos ociosos e gerenciamento de dados distribuídos.

3 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo serão detalhadas as técnicas, teorias e tecnologias utilizadas na proposta e as razões pelas quais cada uma foi escolhida para compor o sistema.

Primeiramente serão explicadas as Arquiteturas e modelos que servem como base para o sistema. São elas, arquitetura *Event-drive*, *Micro Services*, Sistema de Controle e REST. Entender seus funcionamentos e benefícios é fundamental para conseguir enxergar todas as possibilidades da proposta.

A segunda parte do capítulo explica mais sobre os protocolos e topologias de rede utilizados na proposta, realizando um estudo sobre estes de forma a detalhar seu funcionamento e benefícios.

A terceira etapa da fundamentação teórica é a explicação sobre as tecnologias e bibliotecas utilizadas na proposta.

3.1 ARQUITETURAS E MODELOS

3.1.1 Arquitetura de sistema de controle

A estrutura composta por sensores, atuadores, controladores e interface de controle é a arquitetura básica de um Sistema de controle (HARWELL, 2012). Os sensores medem qualidades físicas, como a temperatura e converte esta informação em um sinal elétrico. Os dispositivos atuadores agem de acordo com esta informação. A interface de controle, é onde toda informação e comandos sobre o sistema são apresentados ao usuário. Por fim, o controlador é onde a lógica é processada. O controlador examina as entradas fornecidas pelo operador e pelos sensores, e envia sinais aos atuadores (HARWELL, 2012).

As funcionalidades básicas de um sistema de controle se mantiveram intactas ao longo dos anos, porém a arquitetura para se alcançar estas funcionalidades vem mudando. Os dispositivos de cada categoria estão evoluindo, assim como qualquer tecnologia, tanto em termos de hardware, quanto na lógica do dispositivo. As redes wireless estão substituindo os fios e pontos de E/S(entradas e saídas), enquanto os protocolos de internet estão conectando máquinas e dispositivos, permitindo o controle remoto. Essas novas tendências estão causando uma rápida mudança nos sensores, atuadores, controladores lógicos e interface de operação (HARWELL, 2012).

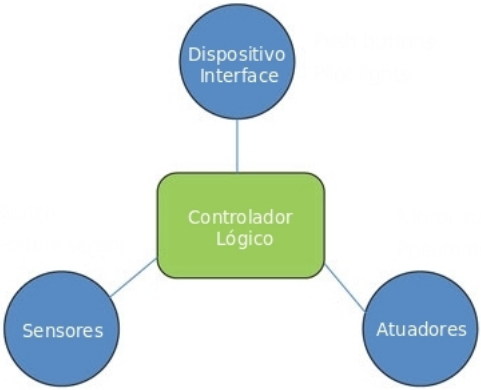

Figura 3 – Componentes Básicos de um Sistema de Controle

Fonte: Fonte Própria

O Sistema de controle, foi inicialmente estabelecido para automação industrial, e utilizava o *Programmable Logic Controller* (PLC), como controlador lógico, este teve seu uso iniciado no começo da década de 70, e se tornou a opção mais utilizada para controle de manufatura (JACK, 2010). O PLC é um dispositivo eletrônico designado especificamente para o controle de máquinas e processos, utilizando memória programável para armazenar instruções e executar funções específicas que incluem o controle de ligar e desligar,*timing*, contador, aritmética e manipulação de dados (AUTOMATION; INSTITUTE, 1995). Porém nos últimos anos novas tecnologias surgiram, como SBC (*single boards computers*) e os micro-controladores, ambos possibilitam a integração com componentes que se ligam ao "mundo real" através das GPIOs e Redes de comunicação.

Em uma casa inteligente, a residência é monitorada de forma contínua, por enviar diversos dados e informações capturadas por sensores, como temperatura e fumaça (RATHORE et al., 2015). Atuadores também são usados a todo momento, e sua ativação pode depender tanto de uma lógica, que recebe como entrada os dados dos sensores ou então por comandos realizados diretamente pelos moradores.

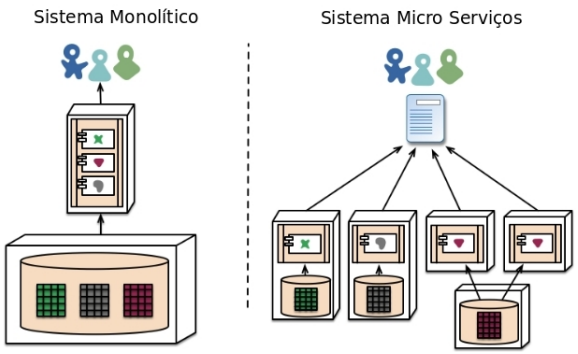

Figura 4 – Comparação entre sistema monolítico e de micro serviços

Fonte: martinfowler.com/articles/microservices

3.1.2 Arquitetura Micro Serviço

O termo Arquitetura de Micro Serviços ou em inglês *Microservices*, surgiu nos últimos anos para descrever um jeito particular de projetar aplicações que contenham serviços independentes uns dos outros (FOWLER, 2014). Não existe uma definição absoluta para esta arquitetura, podendo variar entre autores, porém algumas características estão constantemente presentes, algumas delas são:

- Componentização via serviço: A componentização do código é feita por serviço. Cada serviço é responsável por uma tarefa em específico, e este serviço vai ser componentizado separado de todos os outros. isto permite a sua implantação (*deploy*) separadamente (FOWLER, 2014).

- Organizado por capacidade de negócio: Diferente de outras arquiteturas, onde o código é separado por área, onde existe equipe de *back-end*, equipe de *front-end* e equipe de banco de dados, onde até mesmo para pequenos projetos, a dificuldade de comunicação acarreta em um maior tempo de transição entre as etapas, na utilização de *microservices*, as equipes são divididas por serviço, onde em cada equipe terá pelo menos um membro responsável por cada etapa (FOWLER, 2014).

- *Endpoints* inteligentes, *pipes* burros: Aplicações desenvolvidas

com *microservice* buscam ao máximo ser desacopladas e concisas. Elas possuem seu próprio domínio lógico, e atuam mais como filtros do Unix clássico, recebendo uma requisição, aplicando a lógica e produzindo uma resposta, utilizando protocolos REST, evitando protocolos complexos. Uma segunda abordagem é o envio de mensagem através de um barramento simples, um roteador de mensagem sem nenhuma outra função, a complexidade mora no envio e recebimento das mensagens, o *endpoint* inteligente (FOWLER, 2014).

- Projetados para erros: Uma das consequências da arquitetura *microservice* é a tolerância ao erro. Como os serviços podem falhar a qualquer momento, é importante que o sistema esteja apto a detectar e restaurar o serviço o mais rápido possível. Aplicações *microservice* levam o monitoramento em tempo real de seu funcionamento muito a sério, fazendo verificações de elementos estruturais como métricas relevantes ao negócio (FOWLER, 2014).

- Gerenciamento de dados descentralizado: Significa dizer que o modelo conceitual do mundo será diferente entre o sistemas. Os serviços realizados serão alocados em diferentes lugares e condições, os dados coletados estarão melhores representados levando em conta tais característica (FOWLER, 2014).

- Design incremental: Com a divisão do código em serviços, é fundamental que cada serviço seja escrito de forma objetiva e o mais simples possível, facilitando seu entendimento e manutenção (FOWLER, 2014).

A aplicação de casa inteligente proposta nesta pesquisa, utilizará de algumas características fornecidas pela arquitetura de micro serviços, seus serviços podem e devem ser facilmente componentizados, de modo que cada serviço corresponda a uma atividade possível de ser realizada, como por exemplo abrir uma persiana, regular a temperatura do chuveiro ou ativar o sensor de presença.

O uso da arquitetura de micro serviços trará diversos benefícios à aplicação. Novos serviços poderão ser adicionados sem precisar alterar nenhum código dos outros. Um nodo poderá enviar uma requisição a outro nodo pedindo por um serviço em específico, facilitando a comunicação entre os nodos. E ainda, sempre que um nodo encontrar uma falha ele irá procurar restabelecer seus serviços o mais rápido possível.

A descentralização do banco de dados também trará benefícios. Como cada nodo do sistema receberá uma grande quantidade de in-

formações de seus sensores, armazená-las todas em um banco de dados central iria causar um intenso tráfego nas redes de comunicação, e também um grande uso de processamento do nodo central, que já é responsável por diversas tarefas. Porém, como o limite de armazenamento de dados dos nodos é pequeno por se tratar de dispositivos embarcados, é necessário que em horários de menor utilização do sistema ocorra uma sincronização com o banco de dados central, para que os dados dos nodos sejam então descartados para um novo dia de coleta.

3.1.3 Arquitetura *Event-Driven*

Aqruitetura baseada em eventos, sendo um evento uma ocorrência que acontece dentro ou fora de uma aplicação, podendo significar um problema no sistema, uma ação de usuário, um comando ou uma atividade registrada por sensores. Um evento é dividido em header e body, no *header* são armazenados meta dados, como, nome, *timestamp*, *id* e tipo. No *body* é armazenado o conteúdo do evento, uma informação do que de fato ocorreu.

Uma arquitetura orientada a eventos consiste de uma rede de processamento de eventos. Esta, por sua vez, consiste de aplicações ou componentes de aplicações que processam objetos de eventos.

A arquitetura *event-driven* (EDA), se resume em criar eventos para depois, a partir de seu conteúdo, executar uma tarefa, para isso existe os '*event creators*' e os '*event consumers*'. O criador, no qual é a fonte do evento, apenas sabe que o evento ocorreu. *Consumers* são as entidades que precisam saber que o evento ocorreu, eles então avaliam seu conteúdo e realizam a devida tarefa.

Produtores (*Producers*)

Um produtor de eventos é um componente de software responsável por detectar eventos no ambiente. Exemplo, um sensor, que fica periodicamente lendo os dados de um transdutor térmico, pode por exemplo, emitir um evento para um dispositivo computacional.

Canal (*Channel*)

Um canal pode ser entendido como um meio de transmissão de uma notificação de um componente da rede de processamento de eventos para o outro. Um canal pode receber mais de um tipo de notificação, como por exemplo, "Pedido Aprovado" e ao mesmo tempo, "Pedido Rejeitado". Canais podem receber eventos de múltiplos produtores e torná-los disponíveis para apenas um consumidor.

Consumidor (*Consumer*)

Um consumidor é um agente da rede de processamento de eventos que recebe os eventos, e a partir das informações recebidas ele toma uma decisão acerca do que fazer. Consumidores também são conhecidos como "*event handlers*", "*listeners*" ou ainda "*responders*".

Intermediário (*Intermediary*)

Redes de processamento de eventos podem ser intermediadas por canais apenas ou por canais e agentes de processamento de eventos.

Um canal é entendido como um intermediário de baixo nível, pois, suas funções são limitadas a tarefas relacionadas à comunicação apenas (MICHELSON, 1995).

Uma característica bastante relevante desta arquitetura é que ela é fracamente acoplada, e altamente distribuída, ou seja, a execução da tarefa pelo receptor do evento, independe de quem a enviou, não há a necessidade de existir memória compartilhada. Arquiteturas *event-driven* são usadas de forma mais adequada em sistemas com fluxo assíncrono de tarefas e informações (MICHELSON, 1995).

3.1.4 Arquitetura REST

Conforme a definição de (FIELDING, 2000), REST, que significa *Representational State Transfer* (ou Transferência de Estado Representativo), é um estilo de arquitetura de software para sistemas distribuídos. Fielding ainda afirma que o estilo REST é uma abstração dos elementos arquiteturais de um sistema multimídia distribuído (SANTOS, 2014), são eles:

- Elementos de dados: são classificados como os recursos, o metadado destes recursos e seus identificadores (URIs) e as suas representações que podem ser um elemento HTML, XML, Imagens e etc.
- Conectores: são os vários tipos de atores que utilizam o REST para encapsular as atividades de acesso aos recursos e a transferência de uma representação de um recurso.
- Componentes: são classificados pelos seus papéis na ação de uma aplicação, os tipos de componentes são classificados como servidores de origem, *gateways*, *proxy* e *user agents*. Neste último caso, o exemplo mais comum é um navegador Web.

Em uma arquitetura REST, dados e funcionalidades são considerados recursos e estes recursos são acessados via URIs (*Uniform Re-*

source Identifier), geralmente via links. REST foi concebido baseando-se em HTTP, até pela formação de Fielding que foi um dos principais autores da especificação do HTTP. A arquitetura de um sistema REST geralmente é cliente-servidor e os serviços não possuem estado (*stateless*) (KAMILARIS; PITSILLIDES, 2012).

A essência do REST é a criação de um serviço fracamente acoplado, e que pode ser facilmente reutilizado, implementado sob os protocolos HTTP, URI e outros padrões web (SANTOS, 2014).

O REST ignora os detalhes da implementação de componente e a sintaxe de protocolo com o objetivo de focar nos papéis dos componentes, nas restrições sobre sua interação com outros componentes e na sua interpretação de elementos de dados significantes. Por ser simples e flexível, o REST vai garantir a interoperabilidade e a transação entre a Web e os objetos IoT. A maior qualidade do REST em relação a outros padrões de interoperabilidade para sistemas heterogêneo, é que ao invés de apresentar um conjunto complicado de guias e protocolos para interação entre dispositivos, o REST utiliza simplesmente protocolos HTTP e princípios básicos da Web.

3.2 TOPOLOGIA E PROTOCOLOS DE COMUNICAÇÃO

A comunicação entre os dispositivos do sistema possui peculiaridades diferentes de acordo com sua função e dos dispositivos utilizados, a seguir, será apresentado um estudo sobre as diferentes tecnologias utilizadas para a realização desta comunicação. Um protocolo é uma linguagem usada para permitir que dois ou mais computadores se comuniquem. Assim como acontece no mundo real, se eles não falarem a mesma língua eles não podem se comunicar.

A topologia pode ser representada de várias maneiras e descreve por onde os cabos passam e onde as estações, os nós, roteadores e *gateways* estão localizados. As mais utilizadas e conhecidas são as topologias do tipo estrela, barramento e anel.

3.2.1 Protocolos Internet

A Internet possui 2 protocolos principais na camada de transporte, um protocolo orientado para conexão e o outro não, que são complementares entre si. O protocolo não orientado a conexão é denominado UDP. Sua função é basicamente o transporte de pacotes entre

aplicações, deixando a cargo desta, a criação de protocolos para complementar o UDP. O protocolo orientado a conexão é o TCP. Este por sua vez trata de garantir que o transporte de pacotes seja confiável, através de retransmissões, ordenamento de pacotes e outros serviços que garantem o controle de fluxo e a segurança dos dados transmitidos. Para este trabalho, os dois protocolos são usados em momentos distintos, e portanto sua explicação se faz necessária.

3.2.1.1 TCP/IP

Este protocolo foi desenhado especificamente para prover uma transmissão de *bytes* fim-a-fim confiável em uma interligação de redes não confiável. Uma interligação de redes difere de uma rede singular por apresentar diversas topologias, largura de banda, *delays*, tamanho de pacotes e parâmetros variados. O TCP foi desenvolvido para adaptar as propriedades da interligação de redes de maneira dinâmica e também para ser um opção robusta para situações de falhas. Ele foi definido formalmente pelo RFC 793 em setembro de 1981. Com o passar do tempo, muitas melhorias foram desenvolvidas e diversos erros e inconsistências foram corrigidos. Em outras palavras, o TCP deve prover uma boa performance com a confiabilidade necessária para as aplicações transmitirem dados (TANENBAUM, 2010).

O serviço é realizado tanto por quem envia quanto por quem recebe, criando os chamados '*endpoints*', também conhecidos por *sockets*. Cada *socket* possui um número(endereço), este é formado tanto pelo endereço de ip, quanto pela porta, um número de 16-bits. Para se obter o serviço TCP, uma conexão deve ser estabelecida entre o socket de uma máquina com o socket de outra. Um único socket pode ser usado para realizar múltiplas conexões ao mesmo tempo. Em outras palavras, duas ou mais conexões podem ser estabelecidas em um mesmo socket. O emissor e receptor do protocolo TCP trocam dados em forma de segmentos. Um segmento TCP consiste de um cabeçalho fixo de 20-bytes seguidos por um conjunto de dados (TANENBAUM, 2010).

O software TCP é o responsável por definir o tamanho dos segmentos, podendo distribuir a informação em um ou mais segmentos. Existem duas características limitadoras para o tamanho do segmento. A primeira, todo segmento, incluindo o cabeçalho devem caber no payload IP. Segundo, cada link possui uma Unidade máxima de transferência, ou MTU (*Maximum Transfer Unit*). Cada segmento deve caber

dentro do MTU do emissor e receptor. Em geral, o MTU possuí 1500 bytes e define o limite máximo de um segmento (TANENBAUM, 2010).

O protocolo básico usado nas entidades TCP é o protocolo de Janela Deslizante ou *Sliding Window* utilizando uma janela de tamanho dinâmico. A cada segmento enviado pelo TCP ele inicializa um temporizador visando aguardar um reconhecimento em um tempo limite. Em caso de estouro do temporizador (*timeout*) o TCP retransmite o segmento (TANENBAUM, 2010).

3.2.1.2 UDP/IP

Os Protocolos Internet inclui um protocolo de transporte "connectionless" ou não orientado a conexões, chamado UDP, este, fornece uma maneira das aplicações enviarem dados encapsulados pelo protocolo IP sem que haja estabelecido uma conexão. O UDP foi descrita no RFC 768 e transmite segmentos formados por um cabeçalho de 8 byte, seguido pelo *payload*(TANENBAUM, 2010).

Duas portas são usadas para identificar o *endpoint* do emissor e receptor Quando um pacote UDP chega ao destino, o *payload* é diretamente enviado ao processo atrelado à porta de destino. O protocolo é bem indicado em situações como cliente-servidor quando o cliente envia requisições ao servidor, esperando uma resposta rápida, de modo que se tanto a requisição ou a resposta é perdida, o cliente pode esperar e tentar novamente (TANENBAUM, 2010).

3.2.2 Topologia Estrela

Uma das topologias escolhidas para o projeto, é a topologia estrela, onde todas as estações estão conectadas a um dispositivo concentrador, um *switch*, por exemplo (WIRTH, 2009). Todos os nodos devem obrigatoriamente transmitir os dados a um nodo central, no caso da proposta, este nodo será definido como o servidor central. Este nodo tem o controle de supervisionar o sistema, sendo que, através dele todos os outros nodos podem se comunicar entre si. Toda informação enviada de um nodo para outro, passa primeiro ao dispositivo que fica no centro da estrela.

O servidor central, além de gerenciar a comunicação entre os nodos, ficará responsável por armazenar os estados dos nodos, de modo a controlar a atividade de cada um. Deste modo ele consegue alocar

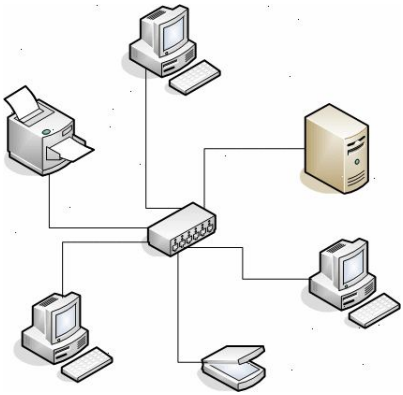

Figura 5 – Topologia Estrela

Fonte: http://o2e.replega.com

tarefas para os nodos que não estejam realizando nenhum tarefa, balanceando a carga de processamento do sistema.

A topologia em estrela é encontrada na maioria dos projetos de redes de computadores e utiliza em larga escala o par trançado como o meio de transmissão (WIRTH, 2009). Uma das vantagens da topologia em estrela, é que a existência de um problema em um dos nodos, não compromete a comunicação entre os outros nodos, e é facilmente identificada. Em uma smart-home, é imprescindível que um defeito em um nodo não comprometa o funcionamento do resto da casa.

3.2.3 Protocolos IoT

Os inúmeros protocolos existentes nos levaram a efetuar um estudo preliminar apresentado em (ROTTA; DANTAS, 2017), o qual nesse capítulo apresentamos de uma forma mais detalhada. Iremos utilizar como modelo de referência, o modelo TCP/IP, usado para os protocolos na Internet. Pelo fato da Internet das Coisas ter sido baseada a princípio na internet, o modelo TCP/IP consegue descrever com naturalidade as camadas utilizadas na mesma. Como exemplificado na tabela abaixo, o modelo é dividido em 4 camadas, Rede(interface com rede), Internet, Transporte e Aplicação (TANENBAUM, 2010).

Camada de Rede (Interface com rede) A Camada de rede é definida como física, ou seja, corresponde às tecnologias usadas para troca

Tabela 2 – Comparativo entre camadas TCP/IP e IP IoT

Camadas	Internet TCP/IP	Ip IoT
Aplicação	HTTP/HTTPS/etc..	CoAP, MQTT
Transporte	TCP/UDP	UDP
Internet	IPv4/IPv6	IPv6/LoWPAN
Interface com rede	IEEE 803.2 / 802.11	IEEE 802.11 / 802.15

de pacotes. No caso da IoT, a comunicação na camada de rede se baseia em comunicação sem fio. Diversos padrões foram especificados para esta camada tais como as séries IEEE 802.11 e 802.15 (SUTARIA; GOVINDACHARI, 2013). Representa a primeira e segunda camada do padrão OSI.

Camada Internet A tarefa da camada inter-redes é entregar pacotes onde eles são necessários (TANENBAUM, 2010). A idéia dos protocolos desta camada é endereçar, entregar e evitar congestionamentos. Representa a terceira camada do padrão OSI.

Camada Transporte No modelo TCP/IP, a camada localizada acima da inter-redes é chamada "camada de transporte". A finalidade desta é permitir que as mensagens partam dos *hosts* de origem ao de destino, mantendo uma conversação (TANENBAUM, 2010). Representa a quarta camada do padrão OSI.

Camada Aplicação Protocolos de serviço específico de comunicação de dados em um nível processo-a-processo. O protocolo de aplicação mais utilizado para prover serviços web é o HTTP, porém possui muita complexidade computacional e consumo de energia elevado para os dispositivos IoT. Representa a quinta, a sexta e a sétima camada do padrão OSI.

3.2.3.1 Padrão IEEE 802.15.4

O grupo da IEEE 802.15 específica uma variedade de '*wireless personal area network*' (WPANs), entre eles Bluetooth, UWB, BAN, e diversas outras. O padrão estudado neste trabalho - 802.15.4 - representa o maior padrão para *low-data-rate* WPANs (FRENZEL, 2013). O principal objetivo do 802.15.4 é prover uma base para que outros protocolos possam ser adicionados nas camadas superiores (transporte, internet e aplicação).

Referido padrão de protocolo atua na camada de rede no modelo TCP/IP, enquanto que as camadas físicas e enlace atuam no modelo

OSI. Assim, a camada física é responsável pela frequência, consumo de energia, modulação e a camada de enlace define o formato do dado (FRENZEL, 2013). O IEEE 802.15 é adotado por diversos grupos que atuam no desenvolvimento de objetos e protocolos IoT, como Zigbee Alliance e Thread Group. Este padrão define protocolos e regras de interconexões para comunicação de dados entre dispositivos utilizando baixa taxa de transferência, baixo consumo e baixa complexidade, utilizando rádio frequência de curto alcance.

3.2.3.1.1 Zigbee

Zigbee é um conjunto de especificações desenvolvido pela Zigbee Alliance para utilização em *smart-home* e IoT, que define as camadas subsequentes às camadas estabelecidas pelo IEEE 802.15.4, oferecendo serviços de segurança, tolerância a erros e conexão de novos dispositivos. Atualmente, é a tecnologia mais utilizada que faz uso do padrão (FRENZEL, 2013). O Zigbee abrange as camadas referentes a internet, transporte e de aplicação do modelo TCP/IP.

3.2.3.2 6LoWPAN

Protocolo de permissão para que pacotes IPv6 sejam transmitidos em redes de baixo consumo energético. Seu conceito inicial era permitir que os protocolos de internet se estendessem para todos os tipos de dispositivos (MULLIGAN, 2007). Referido protocolo segue os padrões definidos no RFC 4944 e RFC 6282, nos quais especificam transmissões IPv6 e sua compressão para camadas de rede IEEE 802.15.4. Sua principal tarefa é comprimir o cabeçalho TCP/IP e também a mensagem enviada, diminuindo o custo de transmissão. Cabeçalhos TCP/IP possuem 128 bytes enquanto IPv6 apenas 40 bytes. Outras tarefas importantes do 6LoWPAN são a fragmentação e reagrupamento de pacotes, assim como a sua distribuição dentro da rede.

3.2.3.3 Camada de Aplicação

Serão analisado dois protocolos emergentes no mundo da Internet das Coisas. Ambos apresentam características de baixo consumo, por possuírem mensagens pequenas, gerenciador de mensagens e pequeno *overhead*, ideal para a utilização em dispositivos embarcados (STANS-

BERRY, 2015). O primeiro protocolo data de 1999, *Message Queuing Telemetry Transport* (MQTT), apesar da idade, seu uso vem ganhando força com o advento da IoT. Enquanto que o segundo, *Constrained Application Protocol* (CoAP), data de 2014 e foi desenvolvido especificamente para IoT.

3.2.3.3.1 MQTT

Inventado pela IBM para comunicação entre satélites e equipamentos de extração de óleo, possui confiabilidade e baixo consumo como requisitos de sua implementação, o que o torna grande candidato a ser aplicado nas redes IoT. Utiliza o modelo *"publish/subscribe"* e necessita de um *broker* MQTT para gerenciar e rotacionar as mensagens dentro da rede. É descrito como um protocolo de comunicação *many-to-many*. Uma das grandes vantagens do MQTT é a eficiência energética do modelo '*pub/sub*', que também escala muito bem. Por utilizar o protocolo TCP, o MQTT já vem munido de segurança na rede. A dependência de um *Broker* e a utilização do TCP também pode ser um empecilho, pois ambos necessitam de um certo poder computacional para funcionar, impossibilitando o uso MQTT em dispositivos mais simples. Por necessitar de um *broaker*, este protocolo se torna uma boa opção para comunicação remota/cloud, pelo fato do servidor *cloud* atuar como o *broaker* entre o dispositivo IoT e outros serviços.

3.2.3.3.2 CoAP

Protocolo que utiliza o modelo 'cliente/servidor', o qual disponibiliza interação '*request/response*' um-para-um, podendo também suportar *multi-cast*. Diferente do MQTT, o CoAP surgiu para suprir a necessidade em protocolos IoT, foi desenvolvido para interoperar com HTTP e com arquiteturas RESTful, através de simples *proxies*, tornando-se compatível com a Internet. Por utilizar o protocolo UDP, o CoAP apresenta menor consumo computacional e energético. Seu uso permite um menor tempo de resposta quando acionado, pois mantém uma conexão ativa entre nodos. Este protocolo é mais indicado para envio de comandos para nodos locais, por se tratar de uma arquitetura semelhante a HTTP. Também é mais utilizado em dispositivos com menos recursos computacionais, por conta das características explicadas acima.

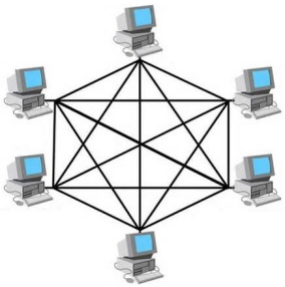

Figura 6 – Topologia Mesh

Fonte: http://www.networking-basics.net/mesh-topology/

3.2.4 Topologia em Malha

Redes *mesh* (redes em malha sem fio) são redes com topologia dinâmica, variável e de crescimento orgânico, constituídas por nós cuja comunicação, no nível físico, é feita através de variantes dos padrões IEEE 802.11 e 802.16, e cujo roteamento é dinâmico (WIRTH, 2009). As redes *mesh* invertem o paradigma de usar uma rede cabeada para a espinha dorsal (*backbone*) da rede, e acesso sem fio na última milha. O *backbone* de uma rede *mesh* é sem fio, e o acesso dos nós clientes pode ser com ou sem fio. Como os nós do *backbone* deste tipo de rede têm localização fixa, estes podem facilmente ser alimentados, não possuindo, desta forma, limitação de energia, eliminando, por conseguinte, muitas das restrições das redes *ad-hoc* (WIRTH, 2009). A proposta deste trabalho utiliza a topologia *mesh* para formar a rede entre os objetos IoT.

3.3 TECNOLOGIAS E BIBLIOTECAS

Para o desenvolvimento do sistema será necessário a utilização de diversas tecnologias e bibliotecas, estas oferecem uma base para que para aplicar e agilizar a implementação dos conceitos explicados até aqui. O motivo da seleção de cada tecnologia assim como suas funcionalidades serão explicados nesta seção.

3.3.1 JavaScript

Considerando os fatos levantados, a linguagem escolhida para o desenvolvimento da proposta foi Javascript, interpretada pelo motor Node.js. O JavaScript foi projetado em uma arquitetura orientada a eventos. Essa arquitetura é perfeita para lidar com entradas e saídas, exatamente o comportamento esperado da proposta, onde comandos e sensores são os principais responsáveis por ativar alguma ação das aplicações. Outro grande benefício é o baixo consumo energético da CPU, extremamente importante para dispositivos embarcados (JSCRAMBLER, 2015). Possuí também uma enorme e crescente comunidade, existindo muito material de aprendizado disponível, assim como também *plugins* e *frameworks* para Raspberry Pi, sistemas distribuídos, servidores HTTP, entre outros (JSCRAMBLER, 2015).

JavaScript é uma linguagem leve, interpretada e baseada em objetos com funções de primeira classe.

Concebida para ser utilizada como scripts de página web, ganhou espaço em outras áreas sem relação alguma com *browsers*, como é o caso do node.js ou do banco de dados CouchDB. É uma linguagem de script baseada em protótipo, multi-paradigma, suportando estilos de programação orientado a objetos, imperativo e funcional.

Segundo Michael Richmond, o JavaScript foi inventado para um objetivo em particular, tratar os casos quando alguém começa a digitar em algum input no *browser* e você não quer que o resto da página trave. A linguagem possui esta semântica, onde você pode escrever um código em JavaScript, e ele é executado automaticamente quando o *input* é alterado. Este é o modelo *event loop* (ROWINSKI, 2016).

O modelo de "*event loop*"em JavaScript, consiste na manipulação de E/S realizada através de eventos e *callbacks* diferentemente de muitas outras linguagens, as E/S nunca são bloqueadas. Então quando uma aplicação está esperando por uma consulta do banco de dados retornar ou uma requisição HTTP assíncrona, este ainda pode processar outras coisas, como as ações do usuário.

3.3.2 Node.js

O Node é um interpretador de código JavaScript que funciona do lado do servidor. Ele utiliza um modelo orientado a eventos não bloqueante de entradas/saídas que o torna leve e eficiente, o Node.js conta com um gerenciador de pacotes chamado NPM, atualmente é o

Figura 7 – Funcionamento não bloqueante do Node.js

Fonte: http://www.imasters.com.br

maior repositório de bibliotecas *open source* do mundo.

O Node.js trabalha com assincronismo. Ele permite que você desenvolva uma aplicação orientada a eventos, tudo isso graças ao *Event-loop*. O *event-loop* é um gerenciador de eventos, é formado por duas bibliotecas nativas da linguagem C, libev e libeio, e são responsáveis por prover a funcionalidade de E/S assíncrona para o Node.js.

Ele é de fato um loop infinito, que a cada iteração verifica se um novo evento foi adicionado à lista de eventos, que são adicionados quando são emitidos durante as emissões de eventos na aplicação, conforme é mostrado na figura 7. O *EventEmitter*, é o responsável por emitir eventos, e a maioria das bibliotecas do Node.js utilizam esse módulo para gerar suas funcionalidades de *emit* e *listen* de eventos.

O *event-loop*, permite executar uma lista de tarefas sem precisar esperar o término de cada tarefa para começar outra. Pode receber

e responder a eventos e então esperar por *callbacks*, que são ativados sempre que um evento for completado. Isso significa que podemos responder a eventos à medida que eles ocorrem, operando diversas tarefas simultaneamente, à medida que elas aparecem. Diversos dispositivos podem inclusive responder ao mesmo evento, e isso funciona muito bem com IoT e *smart-homes* (KAMILARIS; PITSILLIDES, 2012).

Além do *event-loop*, o Node.js possui um *pool* de outras *threads* não bloqueantes, a *thread* principal e as outras *threads* de *background*, se comunicam via filas, que representam as tarefas a serem cumpridas, quando uma tarefa de I/O é requisitada, ela vai ser endereçada aos *workers* de *background*. Estes então anunciam a *thread* principal que a tarefa foi completada chamando uma função de *callback*. Cada instância do Node.js é representada por um único processo, permitindo assim que um *hardware multi-core* consiga realizar outras tarefas não relacionadas ao Node.js ao mesmo tempo (KAMILARIS; PITSILLIDES, 2012).

O Node foi projetado para construir aplicações que podem suportar diversas requisições de forma concorrente. A cada requisição um *callback* é acionado, porém se não existe nenhum trabalho a ser realizada, o node entrará em modo *stand-by*. Esse paradigma é um contraste dos modelos de concorrência mais comuns utilizados nos dias de hoje, o uso de *threads*. Modelos de *threads* são difíceis de usar corretamente e de corrigir erros, causando na maior parte das vezes aplicações com escalabilidade limitada. Com o Node.js, a ocorrência de *dead-lock* é impossível, e isto para um sistema de casa inteligente é fundamental, pois garante o funcionamento contínuo do sistema. Quase nenhuma função do Node utiliza diretamente entradas e saídas, portanto o processo nunca é bloqueado (ROWINSKI, 2016).

A escolha da linguagem JavaScript, junto com o interpretador, se encaixam perfeitamente nas necessidades de uma casa inteligente, podendo suprir a demanda de múltiplas tarefas ao mesmo tempo, além de ter uma comunidade ativa e receptiva e uma repositório repleto de bibliotecas *open-source* que podem ser aproveitados.

3.3.3 Bibliotecas

As bibliotecas escolhidas para o protótipo representam conceitos já esclarecidos anteriormente, como micro serviços, sistema de controle e *framework API REST* para web

3.3.3.1 Cylon.js

Cylon.js é um *framework* JavaScript para robótica e Internet das coisas. Torna a comunicação e comandos para diferentes dispositivos mais simples. O Cylon.js possui um sistema que permite grande compatibilidade em conexão com hardwares. O *framework* suporta um grande número de dispositivos como Arduino, Raspberry PI, Intel Galileo além de mais de 30 outros dispositivos (HYBRIDGROUP, 2015).

A biblioteca contém um conjunto de instruções que permite a manipulação de todas as funções de dispositivos como Raspberry Pi, podendo controlar todas as entradas e saídas do hardwares através da linguagem JavaScript (HYBRIDGROUP, 2015).

3.3.3.2 Express.js

Express.js é um *framework* web, rápido e minimalista. Utilizado para construção de aplicações web que utilizem Node.js, utilizando poucos recursos porém com um conjunto robusto de recursos para aplicativos móveis e web (EXPRESSJS, 2013). O Express.js utiliza métodos de roteamento e *middlewares* para gerenciar as requisições HTTP. Um método de roteamento é derivado a partir de um dos métodos HTTP, e é anexado a uma instância da classe Express (EXPRESSJS, 2013).

3.3.3.3 Koa.js

Assim como o express o Koa.js é também um *framework* web, desenvolvido por parte da equipe do Express.js, que tem como principal diferença o tamanho do sistema, apenas 550 linhas de código. Ainda assim, o Koa.js vem com um conjunto de métodos essenciais para uma aplicação web, como redirecionamentos, suporte a *proxy* e etc. Facilitando e acelerando o desenvolvimento, juntamente ao controle granular sobre sua aplicação para Node.

4 SMART-COMM

A partir das pesquisas realizadas durante o embasamento teórico e análise do estado da arte foi desenvolvida uma proposta, apresentada neste capítulo, denominada Smart-Comm . Um sistema de comunicação entre dispositivos em uma casa inteligente utilizando meios heterogêneos , oferecendo uma base para a realização de tarefas e captura de informação do ambiente, integrando os diferentes dispositivos da rede a partir de protocolos específicos de acordo com as funcionalidades exigidas a cada um deles.

Essa nova estrutura visa atender as necessidade de automação residencial, além de ser uma base para que diversas aplicações possam utilizar as funcionalidades do sistema de maneira muito simples, por exemplo: uma aplicação de inteligência artificial que utiliza as informações do ambiente para determinar ações nos atuadores.

O sistema visa também atender uma futura necessidade que virá a surgir com o conceito de internet das coisas. Estando pronto para receber e operar tais dispositivos.

4.1 PROPOSTA

O sistema proposto será formada por 4 grupos de dispositivos, sendo eles: servidor central, gateways, dispositivos de controle e objetos IoT. Cada qual será explicado detalhadamente neste capítulo.

4.1.1 Servidor central

Sua principal função é rotear as requisições externas (internet) e internas para os *gateways* adequados, assim como, também coletar as informações destes e enviá-las para nuvem. As requisições geralmente são compostas por um comando responsável por realizar alguma função dentro da casa, como acender uma lâmpada ou abrir uma persiana. A comunicação do servidor com os *gateways* será feita através de requisições HTTP, no modelo cliente/servidor. O servidor central pode ou não fazer o papel de um *gateway*, porém, por ser um dispositivo muito requisitado na rede, é recomendável que sua principal atividade seja as funções de um servidor central.

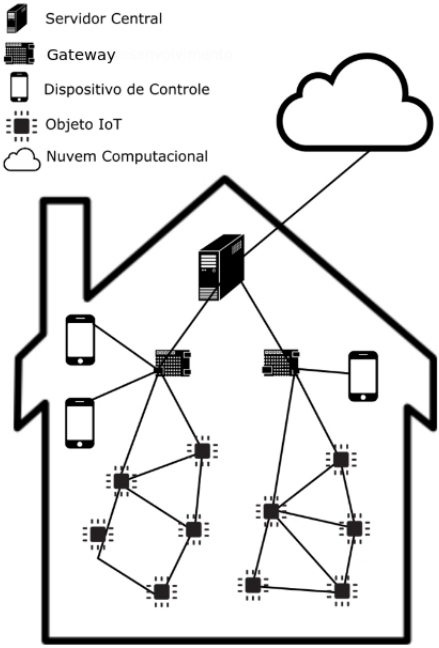

Figura 8 – Esquema da arquitetura da proposta

Fonte: Fonte própria

4.1.2 Gateways

São os atores principais dessa rede. A proposta é possuir uma quantidade de *gateways* que atendam a todos os objetos e sensores da casa e não se sobrecarregue. Um layout sugerido seria uma placa de desenvolvimento(*Single Board Computer*) por cômodo, onde cada placa é responsável por possuir uma tabela de dispositivos IoT conectados a sua rede mesh de internet das coisas, utilizando protocolos específicos para esta rede de baixo consumo energético. Esses dispositivos, assim como os sensores acoplados na placa (a fim de obter informações físicas do ambiente em que a placa está inserida para o conceito de *context-aware*), serão comandados (ativados/desativados ou informação recolhida) através de micro serviços hospedados na placa. Os serviços enviam mensagens para os objetos (via protocolo IoT) ou simplesmente recolhem informações dos sensores através do GPIO da placa ou em

sensores remotos e enviam a informação para o servidor central. Os *gateways* podem possuir diversas funcionalidades, sendo alguns exemplos, *media center*, comandos de voz e fornecimento de conectividade Wi-Fi a dispositivos da casa.

4.1.3 Dispositivo de Controle

O dispositivo de controle é responsável por mostrar ao usuários do sistema as opções de controle da casa, assim como interagir com as mesmas. As opções de controle são determinadas pelos objetos inteligentes dentro da casa , também pelo conjunto de sensores por cômodo. Os dispositivos irão se comunicar com a casa através da internet em um modelo cliente/servidor com o servidor central, ou então diretamente na rede local com as placas de desenvolvimento. Esta segunda comunicação será feita através do protocolo UDP, criando um canal de comunicação com o intuito de aumentar a percepção de tempo real ao usuário, intensificando a velocidade de transmissão entre o dispositivo de controle e o objeto a ser controlado.

4.1.4 Objetos IoT

Podem ser tratados como atuadores e sensores em um sistema de controle. Eles possuem uma peculiaridade que é a moderada capacidade computacional e necessitam de baixo consumo energético. Sua comunicação é feita através de protocolos IoT, na proposta usaremos o Zigbee. Para isso ocorrer, é necessário que o *Gateway* esteja equipado com um *transceiver* com a modulação e frequência de rádio especificada pelo padrão IEEE 802.15.4.

4.2 COMUNICAÇÃO ENTRE DISPOSITIVOS

Uma das características mais importante do sistema proposto é a comunicação entre os diferentes dispositivos. O caminho que um comando segue, desde a chamada do usuário até o objeto IoT podem seguir 3 percursos diferentes, explicados a seguir:

- Comando a um *gateway* local para uma ação de um dispositivo de borda conectado ao mesmo *gateway*. Neste caso o *gateway* recebe do dispositivo de controle um comando para realizar uma

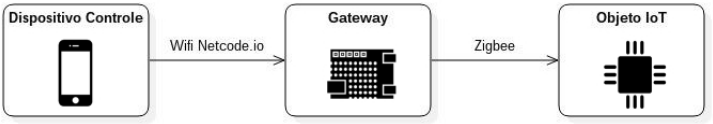

Figura 9 – Representação de comunicação direta de um comando no sistema

Fonte: Fonte própria

ação em um objeto IoT, o *gateway* então, retransmite o comando de forma direta. Ver figura 9.

- Comando a um *gateway* local para uma ação de um dispositivo de borda conectado a outro *gateway*. Neste caso o *gateway* recebe do dispositivo de controle um comando para realizar uma ação em um objeto IoT que não consta em sua rede IoT, desta forma o gateway envia o comando ao servidor central, mapeia qual é o gateway correspondente , retransmite o comando a este, que por sua vez, passa o comando ao objeto IoT. Ver figura 10.

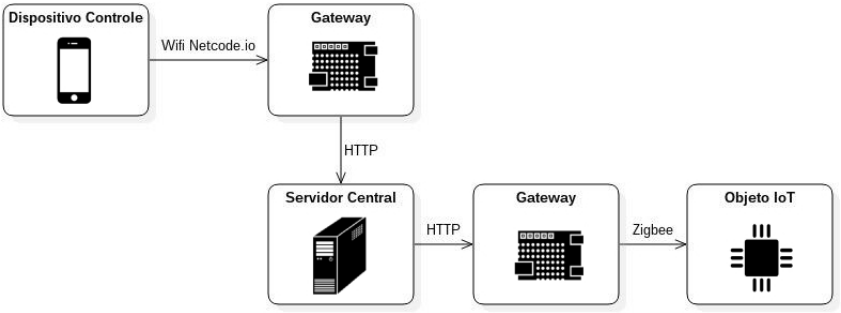

Figura 10 – Representação de comunicação indireta de um comando no sistema

Fonte: Fonte própria

- Comando externo para uma ação de um dispositivo de borda. Neste caso o comando vem de uma entidade externa (nuvem), o servidor central recebe a requisição e mapeia qual é o *gateway* correspondente e retransmite o comando a este, que por sua vez, passa o comando ao objeto IoT. Ver figura 11.

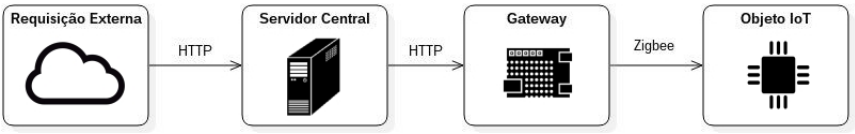

Figura 11 – Representação de comunicação de um comando externo no sistema

Fonte: Fonte própria

4.3 PESQUISAS RELACIONADAS

Esta seção é responsável por mostrar algumas pesquisas relacionadas ao tema proposto, suas diferenças e semelhanças.

4.3.1 *Internet of Things: Ubiquitous Home Control and Monitoring System using Android based Smart Phone*

Nesta pesquisa realizada por (PIYARE, 2013), é proposto um sistema flexível de baixo custo, de monitoramento e controle de uma casa, utilizando um dispositivo embarcado como micro web server, com conectividade ip e controle dos dispositivos utilizando uma aplicação Android.

Esta proposta apresenta muitos pontos em comum com a apresentada neste trabalho, a começar pela arquitetura, que é composta de um servidor utilizando arquitetura REST como uma camada de aplicação para interoperabilidade. A utilização de uma aplicação Android como entidade de controle da casa também é uma característica em comum entre a propostas.

Apesar das semelhanças, as duas propostas apresentam também muitas diferenças. A começar pelo *gateway*, um microcontrolador, diferente de um SBC, com sistema operacional, limitando assim as funcionalidades do gateway, não permitindo por exemplo, uso de micro serviços no gateway e também funcionalidades como *media center*.

A falta de um servidor central também limita o sistema no quesito de processamento e armazenamento local, sendo necessário um servidor na nuvem dedicado a realizar tais tarefas, aumentando o custo mensal do sistema.

A proposta não apresenta um protocolo de comunicação sem fio

específico para com os objetos IoT.

4.3.2 *Design and Implementation of Light-Weight Smart Home Gateway for Social Web of Things*

Pesquisa realizada por (CHUNG et al., 2014), apresenta uma proposta interessante de implementação de um *gateway* para *Social Internet of Things* (SIoT). A arquitetura utilizada nesta proposta, é também baseada em REST e prevê uma integração entre redes sociais e o *gateway*, fazendo uma conexão entre eles. Esta conexão é alcançada através de um servidor web na nuvem.

A proposta tem como finalidade capturar informações de uma casa via sensores, e publicar quando necessário em uma rede social.

A comunicação entre o *gateway* e os sensores é dada através do protocolo Zigbee, assim como nesta pesquisa.

O dispositivo usado como *gateway* na pesquisa de Chung se assemelha mais ao dispositivo proposto na Smart-Comm, trata-se de um DMA-210L *development platform*, um dispositivo dotado de microprocessador. A razão pela utilização de um microprocessador é a quantidade processamento realizado, ao invés de se apoiar totalmente em um servidor na nuvem, o *gateway* realiza processamentos com as informações recebidas dos outros dispositivos do sistema.

Apesar dos objetivos diferentes, as duas propostas dos trabalhos relacionados se assemelham muito, principalmente em termos dos canais e formas de comunicação entre os dispositivos.

4.4 AMBIENTES MODELOS

Nesta seção serão apresentados os cenários e ambientes propostos utilizados para a concepção do protótipo a ser desenvolvido. Características específicas dos cenários e suas possíveis influências no funcionamento do modelo também serão mencionadas. Além do Ambiente Modelo, também será apresentado um ambiente sem a presença do sistema, para evidenciar a diferença entre os cenários, e a importância e os impactos causados pela proposta. Por fim será apresentado um cenário no qual demonstra uma possível escalabilidade do sistema.

4.4.1 Ambiente sem sistema

É o cenário mais comum nos dias de hoje. Nesta categoria se encaixam todos aqueles domicílios e estabelecimentos que não contam com nenhum sistema de automatização. A verificação de informação é feita através de medições realizadas por aparelhos digitais e analógicos, como, termômetro, medidores de umidade, relógios, entre outros. A execução de serviços é realizada manualmente, como por exemplo, abrir janela, trancar uma porta, ajustar temperatura do chuveiro. A verificação dos dados e execução dos serviços podem ser realizadas por qualquer indivíduo com acesso físico a tais medidores e objetos, sendo a única segurança da casa, o dispositivo físico usada para destrancar as portas (chave). Para registrar e salvar as informações é necessário que os dados sejam anotados de forma manual ou digital em algum meio de persistência de dados, um sistema, arquivo ou folha de papel.

4.4.2 Ambiente proposto

O ambiente proposto é aquele no qual o sistema proposto está aplicado, domicílios e estabelecimentos que usufruam das funcionalidades propostas durante este trabalho.

O cenário ideal, seria aquele no qual diversos dispositivos pertencentes ao sistema estariam devidamente ordenados pela casa de acordo com sua funcionalidade:

- Um computador linux atuando como o servidor central do sistema, localizado no centro da casa para melhor conexão com os gateways, hospedando um servidor rodando em Node.js. Conectado aos *Gateways* através de uma conexão sem fio.

- Espalhados pelos cômodos da casa e atuando como *Gateway* na rede, estariam 'Raspberry PI'. A placa rodará uma plataforma de hospedagem de micro serviço, também em um ambiente Node.js em Linux. Munida de diversos serviços implementados para recuperar e ativar os recursos disponíveis na placa e nos dispositivos de borda. Estes serviços utilizam um módulo Node.js para acesso a GPIO chamado Cylon.js. Os *Gateways* se comunicam com os Objetos IoT através de protocolos IoT em uma rede sem fio, como explicado na fundamentação teórica.

- Representando o dispositivo de controle, uma aplicação android

para smartphones, onde o usuário é capaz de realizar os comandos, estes mapeados através dos serviços dos gateways e convertidos em uma interface com o usuário.

- Os Objetos IoT estão representados por qualquer eletrodoméstico, eletrônico ou placas de desenvolvimento e de controle, e devem estar aptos a se comunicarem através de protocolos de comunicação, como por exemplos os protocolos IoT explicados anteriormente.

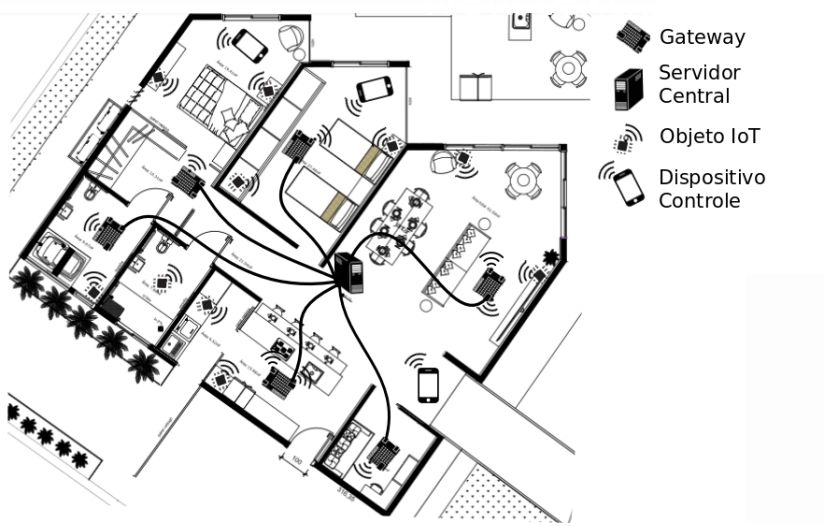

Figura 12 – Representação cenário proposto

Fonte: Fonte própria

A verificação de informação '*context-aware*' é capturada através de sensores localizados nos dispositivos de borda, e enviados ao gateway através de protocolos IoT, ou então capturados por sensores nos próprios *Gateways*. Essas informações são enviadas para o Servidor Central, que realizará a persistência destas através de um banco de dados, ou então, enviada ao dispositivo de controle para que o usuário possa visualizar a informação, ambos envios são através de protocolos internet.

A execução dos serviços é realizada através dos atuadores, localizados também nos dispositivos de borda, e podem realizar as mesmas

tarefas manuais que um morador faz em um ambiente sem sistema, como abrir janelas, regar o jardim e acender uma lâmpada. Os serviços são executados através de uma chamada para um endereço URL correspondente ao micro serviço criado pelo *Gateway*, essa chamada é realizada a partir do dispositivo de controle.

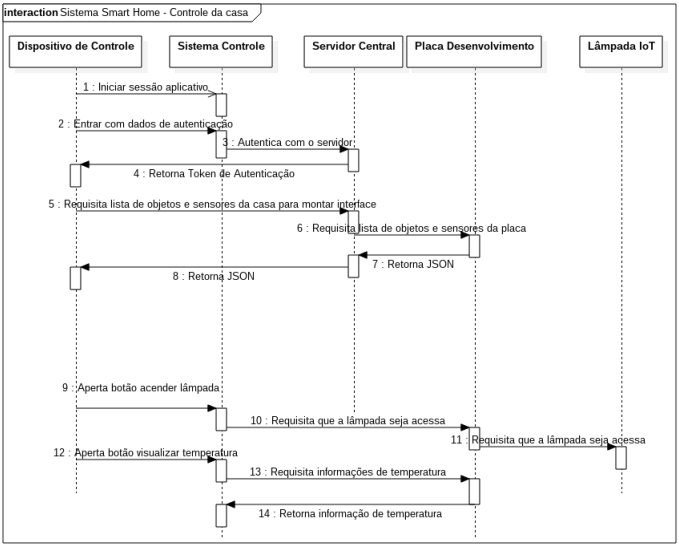

Figura 13 – Diagrama de sequência completo de uma comunicação direta de um comando no sistema

Fonte: Fonte própria

A utilização do sistema no Ambiente Ideal, exige que além do usuário estar conectado a rede local da casa,, o usuário tem que estar credenciado a utilizar tal serviço, para isso o sistema possui um gerenciador de usuários e de autenticação, para prover segurança e proteção aos moradores, contra usuários mal intencionados.

A fim de facilitar o entendimento do ambiente descrito, foi desenvolvido um protótipo funcional do sistema, este engloba algumas das funcionalidades propostas neste cenário, o protótipo será descrito no capítulo seguinte.

4.4.3 Ambiente escalável

Outro ambiente importante a ser mencionado é formado pelo conjunto de diversos "ambientes propostos", como condomínios residenciais, bairros, e até mesmo cidades. Da mesma forma que o Servidor Central gerencia o acesso aos serviços dos gateways e centraliza a informação sobre os serviços e seus endereços, assim como os dados da casa, é possível estabelecer uma nova camada, criando um novo Servidor Central, como por exemplo, um Servidor Central de um condomínio residencial. O servidor é responsável por centralizar as informações de todos os outros servidores e gerenciar seus usuários e seus acessos. Esta estratégia pode ser executada em diversos níveis, essa característica de escalabilidade foi observada durante a elaboração da proposta.

Utilizando o exemplo citado, o condomínio poderia capturar informações relevantes para a sua gestão e manutenção, consumo de energia por residência e consumo de água por residência. O Servidor Central do condomínio tem a responsabilidade de gerenciar e restringir o uso de seus serviços por usuário. Também o Servidor Central de cada residência deve permitir ao servidor do condomínio acesso a estes serviços.

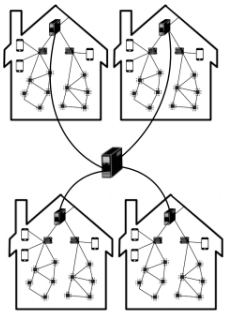

Figura 14 – Representação condomínio residencial

Fonte: Fonte própria

Da mesma maneira que o servidor do condomínio pode acessar dados e informações das residências, ele pode também ordenar comandos aos atuadores nos quais ele tem permissão.

A ideia do ambiente escalável pode ser aplicada a diferentes níveis, onde o Servidor Central do nível mais alto, tem como responsabilidade a centralização e gerência dos serviços e usuários.

5 PROTÓTIPO

Este capítulo aborda as técnicas e estratégias realizadas para a implementação do protótipo, a fim de exemplificar a proposta do sistema e ajudar no seu entendimento.

5.1 VISÃO GERAL

Assim como foi descrito na proposta, as etapas de implementação também podem ser divididas em dispositivos, pelo fato de que cada um possui funções diferentes dentro do sistema e para isso necessitam de tecnologias e abordagens de desenvolvimento diferentes.

O Servidor Central possui uma arquitetura monolítica, reunindo todos os serviços em um único servidor, pelo fato deste dispositivo representar uma entidade centralizadora dentro do sistema. No protótipo o servidor controla o acesso de usuários e mantém as informações sobre os *Gateways* ativos no sistema.

Os *Gateways* possuem uma arquitetura de micro serviços, onde cada serviço é implementado de forma desacoplada no sistema, essa abordagem facilita a inserção e retirada de novos serviços em tempo real, esta estratégia favorece também a tolerância a erros e a componentização dos códigos.

Os objetos IoT no protótipo são representados como scripts de execução, ativados pelos serviço do *Gateway*.

O principal objetivo da construção do protótipo é facilitar a compreensão e entendimento do sistema proposto, assim como também, mostrar que é possível e viável sua implementação.

O protótipo aborda uma série de funcionalidades primordiais para o desenvolvimento do sistema smart-comm apresentado, como:

- Segurança de acesso aos dispositivos, utilizando *tokens* e criptografia para autenticação de usuários: A segurança é fundamental para garantir que os comandos e serviços disponíveis na casa inteligente sejam restritos somente a usuários com autorização, evitando assim usuários mal intencionados e espionagem de dados.

- Roteamento de dispositivos: Necessário para informar aos usuários quais dispositivos estão disponíveis para serem utilizados.

- Gerência de serviços: Permite a inserção e remoção de serviços em tempo real, fundamental para quando o usuário do sistema

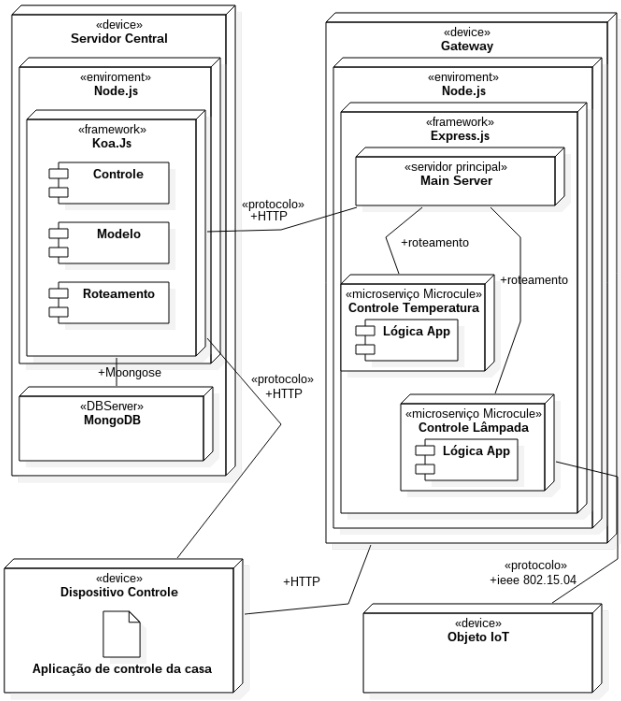

Figura 15 – Diagrama de deployment do Sistema Proposto

Fonte: Fonte própria

decide instalar um novo dispositivo de borda e precisa adicionar um novo serviço no sistema.

- Mapeamento de serviços: Fundamental para que os serviços sejam executados, é necessário antes saber o *endpoint* correto de cada um.

É importante ressaltar que o protótipo não engloba todas as formas de comunicação e funcionalidades propostas, o foco do protótipo é facilitar a concepção idéia.

A implementação do protótipo foca na comunicação e gerência dos dispositivos da casa, por esse motivo, nenhuma interface gráfica foi desenvolvida, e também, o dispositivo de controle será representado por um aplicativo de envio de requisições web, chamado "Postman".

5.2 SERVIDOR CENTRAL

O servidor central, como explicado no capítulo 4, é o responsável por fazer o roteamento das requisições, realizando chamadas aos dispositivos de acordo com a tarefa requisitada. O protótipo irá cobrir a comunicação direta deste com os *Gateways* e com o dispositivo de controle.

A implementação do protótipo conta com diversas funcionalidades essenciais para o funcionamento da proposta. A principal delas é garantir que o usuário seja credenciado a fazer uso do sistema. Para isso o protótipo conta com uma API de gerenciamento de usuário, onde é possível incluir e visualizar os usuários do sistema, para maior segurança a senha é armazenada criptografada em um banco de dados. Após a autenticação, o servidor gera um *token* de conexão, este *token* é utilizado pelo dispositivo de controle para requisitar serviços aos Gateways.

O protótipo também assume a responsabilidade de mapear os serviços oferecidos por todos os *Gateways* conectados ao sistema, fornecendo ao usuário a listagem de todos os *Gateways* e seus respectivos serviços.

O Banco de Dados utilizado foi o MongoDB, porém qualquer banco de dados poderia ser utilizado sem comprometer os objetivos do protótipo.

A configuração dos aplicativos e programas necessários foram feitos através do Vagrant, um programa que oferece uma instalação rápida de um ambiente de desenvolvimento em uma máquina virtual, com todas as dependências necessárias para que a aplicação possa ser executada. Esse ambiente contém um *script* que é executado assim que a máquina virtual é inicializada pela primeira vez, instalando todos os programas e bibliotecas necessários.

Todos os códigos utilizados estão disponíveis nos Anexos deste trabalho.

5.2.1 Descrição do desenvolvimento

A implementação do Servidor Central foi realizada utilizando exclusivamente a linguagem Javascript, a escolha da linguagem foi devida ao fato desta ter sido projetada em uma arquitetura orientada a eventos, perfeita para lidar com entradas e saídas, exatamente o comportamento esperado da proposta. Fez-se o uso do interpretador Node.js, que uti-

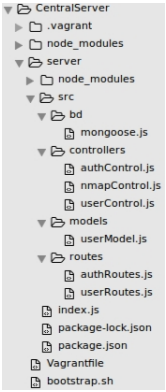

Figura 16 – Estrutura de arquivos - Servidor Central

Fonte: Fonte própria

liza o *event-loop*, um gerenciador de eventos responsável por prover a funcionalidade de E/S assíncrona.

O código consiste em um servidor API monolítico, utilizando como base o framework web "Koa.js". A escolha do framework se deve ao fato deste utilizar a versão mais recente do Node.js, podendo aproveitar todas as funcionalidades existente no interpretador. A estrutura do servidor pode ser divida em arquivos de controle, modelo e roteamento como mostra a figura 14.

De modo a facilitar o entendimento do Servidor Central, a implementação será explicada em módulos, manipulação de usuário e mapeamento de serviços, as funcionalidades existentes dentro do Servidor Central.

5.2.1.1 Manipulação de Usuários

Para que seja possível realizar autenticação de usuários dentro do sistema, é necessário possuir além de um algoritmo de autenticação, um gerenciador de usuários, que permita incluir novos usuários.

Uma API web fornece aos usuários a possibilidade de cadastro e autenticação no sistema através de requisições POST. A API fornece também, a possibilidade de listar os usuários existentes.

O servidor conta com um arquivo de modelo do usuário, “userModel.js”, onde é realizado o esquema do modelo no banco de dados,

dois arquivos de controle, “userControl.js”, onde são realizadas as buscas e manipulação dos dados do usuário, e um arquivo de roteamento para associar os endereços da API com as funções de controle, “userRoutes.js”.

A autenticação do usuário consiste em verificar a existência do nome de usuário e senha, fornecidos através da requisição POST, no banco de dados. Após a verificação, caso a combinação exista, a autenticação retorna um usuário. É gerado um token de acesso utilizando uma chave secreta compartilhada com os *gateways* e com ele é possível realizar requisições aos serviços do sistema.

5.2.1.2 Mapeamento de serviços

O mapeamento de serviços fornece aos usuários uma listagem de todos os gateways disponíveis no sistema e também todos os endpoints dos serviços oferecidos por eles. Esta listagem pode também ser utilizada pelo dispositivo de controle para montar a interface do usuário.

O mapeamento de serviços consiste em realizar uma busca dos *gateways* participantes do sistema, fazendo uma requisição a estes, onde é pedida a listagem de seus serviços. Após coletar a informação de todos os Gateways, a listagem de todos os serviços fica disponível através de uma chamada a API.

Para isso o servidor conta com o arquivo de controle "mapServicesControl.js", este se utiliza do módulo node "*scan-neighbors*", responsável por mapear todos os dispositivos cuja determinada porta está aberta na rede, no caso do protótipo, os *Gateways* ocupam a porta 4000. Após todos os dispositivos mapeados, o Servidor Central faz uma requisição pedindo a listagem de serviços de cada Gateway, montando as informações em uma estrutura de dados.

O arquivo responsável pela rota da API de mapeamento é nomeado "mapRoutes.js".

5.3 GATEWAYS

O desenvolvimento do *Gateway* no protótipo do sistema tem como foco a gerência dos múltiplos serviços dos dispositivos de borda, utilizando a estratégia de micro serviços, além de restringir a utilização destes serviços a usuários não cadastrados.

Os serviços são responsáveis por executar tarefas nos dispositivos

de borda. Estas podem ser inicializadas através de uma chamada de API, onde cada *Gateway* possui um ip na rede. Cada serviço está sendo designado a uma porta diferente.

É possível também listar os serviços de cada *Gateway*, através de outra chamada a API, a listagem apresenta o nome do serviço, sua respectiva porta, descrição e parâmetros. A forma como os serviços foram estruturados servem apenas como exemplo para elucidar o funcionamento dos micro serviços propostos.

Por fim o *Gateway* pode também fazer o papel de um dispositivo de borda, e para isso é necessária algumas configurações extras, exemplificadas no anexo.

5.3.1 Descrição do desenvolvimento

A implementação do *Gateway* foi realizada utilizando exclusivamente a linguagem JavaScript, para se aproveitar dos mesmos benefícios do Servidor Central, também foi utilizado o interpretador Node.js.

O código consiste em um servidor principal, utilizando como base o framework web "Express.js", este é responsável por instanciar os micro servidores utilizando o framework "Microcule". A razão pela qual a escolha do framework do servidor principal do *Gateway* ser diferente do Servidor Central foi devido a sua facilidade de integração com o framework de micro serviços.

Para facilitar o entendimento do *Gateway*, a implementação será explicada em módulos, cada qual, uma funcionalidade diferente dentro do protótipo: micro serviços, listagem de serviços e configuração dispositivo de borda.

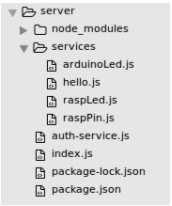

Figura 17 – Estrutura de arquivos - Gateway

Fonte: Fonte própria

5.3.1.1 Micro serviços

A principal função do protótipo do *Gateway* é a gerência de serviços, isso significa instanciar os diferentes serviços e verificar sempre que um novo serviço é incluído ou retirado, ou seja, toda vez que um novo serviço é adicionado ao diretório de serviços (figura 17), o servidor principal sobe uma nova instância de servidor para atender aquele código novo, e toda vez que um serviço é retirado, o servidor remove aquela instância, não permitindo um novo acesso.

O servidor principal do *Gateway* ao ser instanciado verifica o diretório "services", ele então itera por cada arquivo(serviço), criando uma instância de um serviço, utilizando o "Microcule", e então realiza o deploy do serviço em uma nova porta disponível. As portas destinadas aos serviços começam pela 4001 e a cada novo serviço ela é incrementada. O servidor principal do *Gateway* também possui um observador para este mesmo diretório, para que, a cada inclusão de um novo serviço este seja instanciado, e a cada remoção, o serviço seja desligado.

Cada serviço, além do código de execução, possui uma descrição informando as característica do serviço e também possíveis parâmetros, que podem ser enviados ao serviço através do método POST.

Ao instanciar um novo serviço, o servidor inicializa também uma instância do serviço de autenticação, forçando a requisição a passar sempre antes pela autenticação.

A autenticação funciona como no Servidor Central, utilizando *tokens*. Para validar o *token* é utilizada uma chave compartilhada. O *token* é obtido através do Servidor Central.

5.3.1.2 Listagem de serviço

Outra função do protótipo é a listagem de serviços de um Gateway, a listagem pode ser usada tanto pelo Servidor Central, para reunir todos os serviços do sistemas, como pelo dispositivo de controle, para ter acesso aos endereços de cada serviço e oferecer ao usuário uma interface de acesso.

A listagem é obtida salvando os dados de cada serviço no momento em que o servidor principal do *Gateway* itera sobre os arquivos, guardando as informações necessárias em um dicionário.

É possível obter a listagem através de um método POST para a API, utilizando o *token* de acesso para autenticar a requisição.

5.3.1.3 Dispositivo de borda

Entre os micro serviços disponíveis, existem aqueles que utilizam do próprio dispositivo como fornecedor do serviço. O protótipo utiliza um Raspberry Pi como dispositivo *Gateway*, um dos serviços disponíveis, oferece a possibilidade de ativar um PIN arbitrário do dispositivo, passado por parâmetro POST.

Para facilitar esse tipo de controle, utilizamos a biblioteca Cylon.js.

Para ter acesso aos PINS do raspberry é necessário seguir uma série de instruções, essas estarão disponíveis no anexo deste trabalho.

5.4 OBJETO IOT

O protótipo permite duas maneiras distintas de operar um dispositivo de borda (Objeto IoT), a primeira é diretamente pelo micro serviço do gateway, onde cada serviço possui o código no qual realiza uma função do dispositivo, é utilizado para isso a biblioteca Cylon.js.

Outra maneira é pelo comando enviado do gateway através de *transceivers* e estes se comunicarem diretamente com os Objetos IoT. Neste caso o dispositivo de borda, precisa ajustar seu controlador para interpretar o sinal e executar a função desejada, já o *gateway* precisa realizar o envio de sinais elétricos para o *transceiver* de acordo com o protocolo escolhido. A utilização de protocolos IoT são recomendadas nesta segunda abordagem. Dada a complexidade do código, não foi implementado no protótipo o segundo tipo de controle sob os objetos de borda.

5.5 UTILIZAÇÃO DO PROTÓTIPO

Para demonstrar o funcionamento do protótipo, foram criados cenários exemplificando duas funcionalidades disponíveis. As funcionalidades do protótipo são executadas através de requisições, na proposta geral, o responsável por elas é o dispositivo de controle, para o protótipo foi utilizada a aplicação "Postman", e portanto se faz necessário explicar sua interface para melhor entender os casos de uso que serão explicados posteriormente.

A figura 18 apresenta a interface padrão da aplicação "Postman", a explicação dos principais pontos será detalhada a seguir:

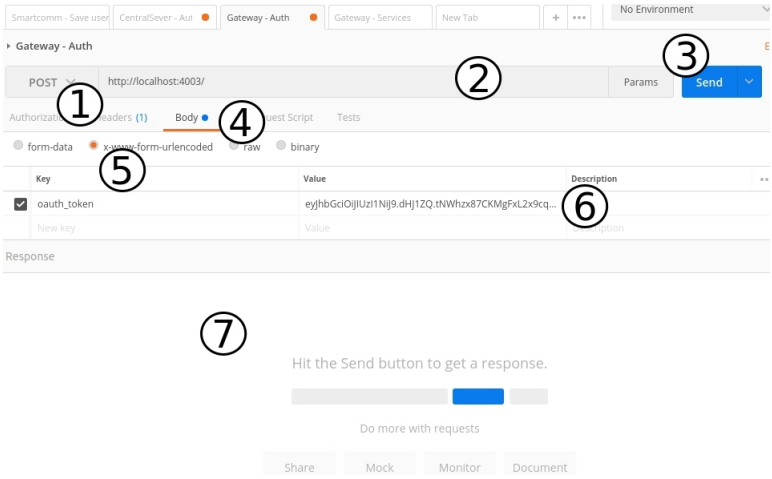

Figura 18 – Interface aplicação Postman

Fonte: Fonte própria

1. O primeiro item corresponde a um *menu dropdown*, contendo o tipo da requisição que se deseja enviar. Neste caso de uso será utilizada apenas a requisição POST.
2. Endereço URL no qual se deseja enviar a requisição.
3. Botão no qual dispara a requisição para a URL desejada.
4. Aba referente ao corpo da requisição a ser enviada, nesta aba é possível configurar os parâmetros da mensagem.
5. "*Radio button*"referente ao tipo de dados que serão enviados.
6. Este item exemplifica os campos onde parâmetro da mensagem podem ser configurados.
7. Corpo da resposta, nesse campo aparece a resposta a requisição emitida pelo servidor da URL enviada.

5.5.1 Criação de usuário e Autenticação nos Sistema

A primeira ação realizada para poder utilizar o sistema protótipo é o cadastro de um novo usuário, sem ele nenhuma funcionalidade

do sistema é acessível. A restrição dos serviços do protótipo a usuários cadastrados é essencial para garantir um controle dos usuários capacitados a exercer comandos dentro da casa inteligente, sem essa restrição qualquer visitante poderia emitir comandos, alguns destes comandos potencialmente perigosos para a segurança dos moradores.

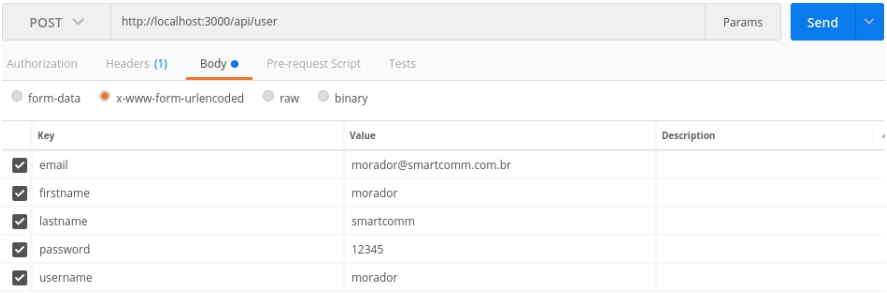

Figura 19 – Requisição POST - Salvar usuário

Fonte: Fonte própria

O cadastro do usuário é feito atrás de uma requisição à API do Servidor Central, considerando que este servidor esteja utilizando o endereço localhost e a porta 3000, é necessário enviar uma requisição POST ao endereço http://localhost:3000/api/user com os seguintes parâmetro, email, firstname, lastname, password e username, conforme mostra a figura 17.

É possível ver a listagem dos usuários cadastrados através de uma chamada GET ao servidor, neste caso, como mostra a figura 18, é possível realizar a chamada pelo próprio navegador web.

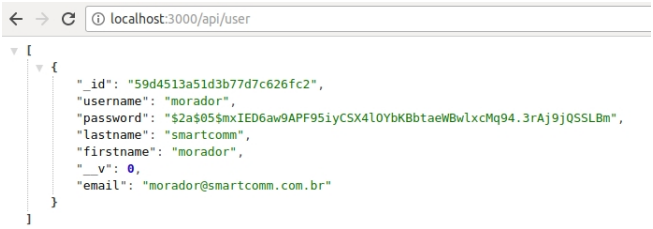

Figura 20 – Requisição GET - visualizar usuários

Fonte: Fonte própria

A partir do usuário cadastrado é possível realizar a autenticação no sistema, ao realizar uma requisição POST ao endereço http://localhost:3000/ap passando como parâmetro no corpo da mensagem o nome de usuário e senha, em caso de ambos os campos serem compatíveis com o de algum usuário cadastrado o sistema retornará o endereço de ip de um *gateway* disponível e o *token* de acesso para realizar chamadas aos serviços oferecidos.

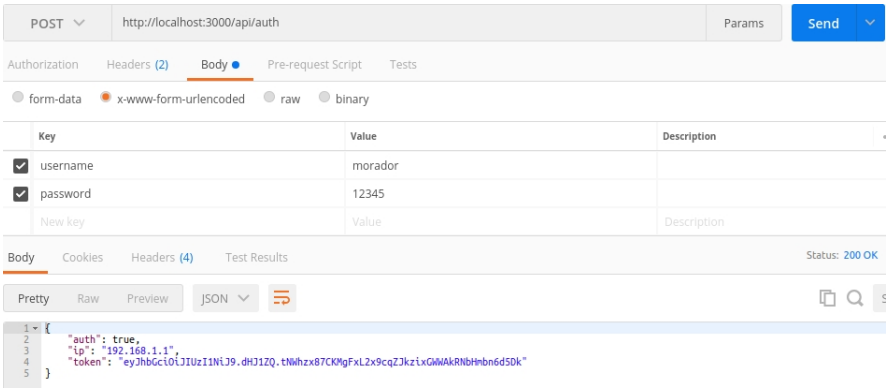

Figura 21 – Requisição POST - Autenticar usuário

Fonte: Fonte própria

5.5.2 Chamada de um micro serviço pelo Gateway

A realização de um comando para um dispositivo de borda é feita a partir de um serviço oferecido pelo *Gateway*, para efetuar a chamada a um serviço, o usuário precisa antes possuir um token de autenticação e uma lista com os endereços para chamada de cada serviço.

Com o *token* de acesso obtido na autenticação do servidor central, o usuário consegue através de uma requisição à api do servidor principal do *Gateway* a listagem de todos os serviços oferecidos por ele, levando em consideração que o endereço de ip do *Gateway* seja 192.168.1.2, é possível ter acesso a um JSON da lista de serviços através de uma requisição POST ao endereço URL http://192.168.1.2:4000/services. A resposta obtida pode ser verificada na figura 20.

Após obter o a porta correspondente ao serviço desejado, e com a posse do *token* de autenticação é possível finalmente requisitar um

Figura 22 – Requisição POST - Listagem de serviço

Fonte: Fonte própria

serviço ao *Gateway*. O serviço usado como exemplo é o "hello.js"cuja porta é a 4004. Na figura 21 podemos verificar a requisição POST ao endereço http://192.168.1.2:4004, utilizando o parâmetro "oauthtoken"para poder realizar a autenticação.

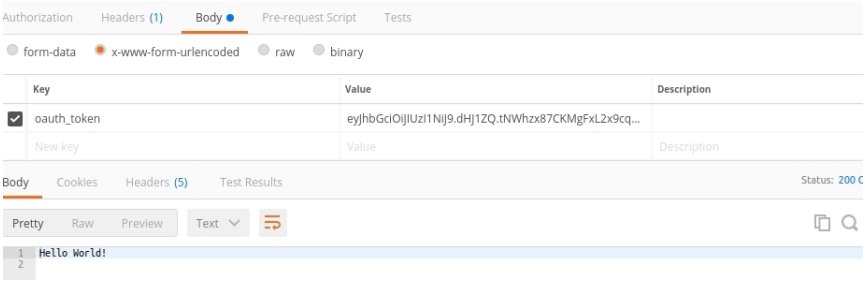

Figura 23 – Requisição POST - Serviço Hello World

Fonte: Fonte própria

6 CONCLUSÃO E TRABALHOS FUTUROS

Smart home e IoT são considerados temas emergentes dentro do cenário acadêmico, e apesar do número crescente de artigos e pesquisas publicadas a cada ano, existe muito ainda a ser definido, como por exemplo, especificações dos protocolos IoT na camada de aplicação. A falta de padrão leva a criação de diferentes vertentes, gerando diversos problemas de interoperabilidade e consistência.

Este trabalho sugere uma proposta de um sistema ubíquo para solucionar diversos desafios existentes hoje em dia com relação a internet das coisas e casas inteligentes, utilizando um conceito heterogêneo de comunicação de modo a relacionar técnicas e arquiteturas computacionais com dispositivos existentes e de baixo custo. Esta visão de relacionamento entre tecnologias e arquiteturas foi possível graças a pesquisa realizada para a fundamentação teórica.

A partir das tecnologias levantadas e analisadas foi possível propor uma arquitetura de um sistema para casas inteligentes, implementando um protótipo funcional do sistema e verificando a viabilidade de sua construção.

O protótipo visou estabelecer a conexão entre os dispositivos do sistema, disponibilizando APIs para troca de mensagens entre os dispositivos, dividindo os serviços entre os *Gateways* e utilizando o Servidor Central como gerenciador dos dispositivos e usuários.

A conexão entre o *Gateway* e o Objeto IoT, prevista para estar no protótipo se mostrou complexa para a realização ainda dentro deste trabalho, devido a necessidade de dispositivos específicos e de configurações e conhecimentos avançados no assunto. A utilização do Protocolo IoT entre *Gateway* e dispositivos de borda passam a ser um item a ser realizados em trabalhos futuros.

Durante o desenvolvimento da protótipo ficou evidente a escalabilidade da arquitetura. Apesar da proposta ter como escopo uma residência, esse escopo pode ser ampliado, devido a facilidade de se compartilhar e acessar os serviços. É aplicável a um trabalho futuro novas maneiras e técnicas de se obter a escalabilidade deste sistema.

Outro tema aberto a trabalhos futuros é o desenvolvimento de um aplicativo para o dispositivo de controle, no protótipo desenvolvido foi utilizado uma aplicação de gerenciamento de requisições, porém a proposta sugere a utilização de um aplicativo em *smartphone* para a realização de requisições e visualizações dos dados da casa.

REFERÊNCIAS

ATZORI, L. The internet of things: A survey. *Computer Networks*, 2010.

AUTOMATION; INSTITUTE, P. *Comparative Strategies for Implementing Embedded Control Systems*. [S.l.], 09 1995.

BISGAARD, J. J.; HEISE, M.; STEFFENSEN, C. Context aware computing for the internet of things: A survey. 2004.

CHUNG, T.-Y. et al. Design and implementation of light-weight smart home gateway for social web of things. *Ubiquitous and Future Networks (ICUFN), 2014 Sixth International Conf*, 2014.

DEY, A. K.; ABOWD, G. D. Towards a better understanding of context and context-awareness. *Proceeding HUC '99 Proceedings of the 1st international symposium on Handheld and Ubiquitous Computing*, 1999.

EXPRESSJS. *Express*. [S.l.]: GitHub, 2013. Http://expressjs.com/.

FIELDING, R. *Architectural Styles and the Design of Network-based Software Architectures*. Tese (Doctor dissertation) — UNIVERSITY OF CALIFORNIA, IRVINE, 2000.

FOWLER, M. *Microservices*. 2014. Https://martinfowler.com/articles/microservices.html.

FRENZEL, L. What's the difference between ieee 802.15.4 and zigbee wireless? *Electronic Design*, 2013. <http://electronicdesign.com/what-s-difference-between/what-s-difference-between-ieee-802154-and-zigbee-wireless>.

HARWELL, R. Integrated hmi and plc -the heart of a "lean automation"solution. *InTech Magazine*, 2012. <https://www.isa.org/standards-publications/isa-publications/intech-magazine/2012/august/cover-story-integrated-hmi-plc/>.

HYBRIDGROUP. *Cylon*. [S.l.]: GitHub, 2015. Http://expressjs.com/.

JACK, H. *Automated Manufacturing Systems with PLCs*. [S.l.]: Lulu.com; 7th edition, 2010.

JSCRAMBLER. *JavaScript: The Perfect Language for the Internet of Things (IoT)*. 2015. Https://blog.jscrambler.com/javascript-the-perfect-language-for-the-internet-of-things-iot/.

KAMILARIS, A.; PITSILLIDES, A. A restful architecture for web-based smart homes using request queues. *Technical Report No. TR-12-5, Department of Computer Science, University of Cyprus*, 2012.

MICHELSON, B. M. *Event-Driven Architecture Overview*. [S.l.], 1995.

MULLIGAN, G. The 6lowpan architecture. *EmNets '07: Proceedings of the 4th workshop on Embedded networked sensors*, 2007. ISSN 978-1-59593-694-3. <http://dl.acm.org/citation.cfm?id=1278992>.

PERERA, C.; ZASLAVSKY, A.; CHRISTEN, P. Context aware computing for the internet of things: A survey. *IEEE COMMUNICATIONS SURVEYS AND TUTORIALS, VOL. 16*, 2014.

PIYARE, R. Internet of things: Ubiquitous home control and monitoring system using android based smart phone. *International Journal of Internet of Things 2013*, 2013.

RATHORE, M. M. et al. Urban planning and building smart cities based on the internet of things using big data analytics. *Computer Networks*, 2015.

ROTTA, G.; DANTAS, M. Um estudo sobre protocolos de comunicação para ambientes de internet das coisas. *Escola Regional de Alto Desempenho*, 2017.

ROWINSKI, D. *How JavaScript Came To Rule The Internet Of Things*. 2016. Https://arc.applause.com/2016/02/23/javascript-node-internet-of-things/.

SANTOS, W. R. dos. *RESTful Web Services e a API JAX-RS*. 2014. Http://www.ricardoluis.com/wp-content/uploads/2015/08/Artigo-WebServices-em-REST.pdf.

STANSBERRY, J. Mqtt and coap: Underlying protocols for the iot. *Electronic Design*, 2015. <http://electronicdesign.com/iot/mqtt-and-coap-underlying-protocols-iot>.

SUTARIA, R.; GOVINDACHARI, R. Understanding the internet of things. *Electronic Design*, 2013. <http://electronicdesign.com/iot/understanding-internet-things>.

TANENBAUM, A. *Computer Networks 5th.* [S.l.]: Pearson, 2010.

THREAD. *Whitepaper, Thread Stack Fundamentals.* [S.l.]: Thread Group, 2015.

VAQUERO, L. M.; RODERO-MERINO, L. Finding your way in the fog: Towards a comprehensive definition of fog computing. *ACM SIGCOMM Computer Communication Review*, 2014.

WEISER, M. Open house. *Review, Interactive Telecommunications Program of New York University.*, 1996.

WEISER, M. The computer for the 21st century. *ACM SIGMOBILE Mobile Computing and Communications Review*, 1999. <http://dl.acm.org/citation.cfm?id=329126>.

WIRTH, A. *Rede de Computadores: Tecnologia e Convergência das Redes.* [S.l.]: Alta Books, 2009.

ANEXO A – Trabalho Publicado

Este capítulo descreve e apresenta o artigo publicado no ERAD-RS, Escola Regional de Alto Desempenho do Rio Grande do Sul. O Artigo foi resultado de pesquisas realizadas na área de protocolos de comunicação IoT, durante o desenvolvimento deste presente trabalho, e visou entender os conceitos e dificuldades envolvendo este assunto. O objetivo do artigo é explicar quais protocolos são mais utilizados e adequados para cada camada de rede, suas vantagens, desvantagens e desafios.

Um Estudo sobre Protocolos de Comunicação para Ambientes de Internet das Coisas

Giovanni Rotta, Andrea Charão, Mario Dantas

11 de dezembro de 2017

RESUMO

Este trabalho apresenta um estudo com foco em protocolos utilizados na abordagem Internet of Things (IoT), utilizando método de pesquisa exploratória baseada em revisão bibliográfica. A necessidade de criação de novos protocolos, surgiu como solução para a grande expansão de objetos IoT e middlewares, que gerou diversos desafios de interoperabilidade e comunicação. O trabalho busca apontar as causas e benefícios desses novos protocolos.
Palavras-chave: Casas Inteligentes. Internet das Coisas. Computação Ubíqua. Context-Aware. Microservice.

A.1 INTRODUÇÃO

Vivenciamos uma transformação na maneira de enxergar a computação. Nos últimos anos o conceito de computação ubíqua ou pervasiva ganhou um grande impulso com a chegada do conceito de "Internet of Things" (Internet das Coisas). Este, impulsionado pelo desenvolvimento de dispositivos embarcados cada vez mais potentes e pelas possibilidades trazidas pela comunicação wireless, vem ganhando cada vez mais espaço entre pesquisadores e indústria.

Dentro deste contexto, este trabalho visa analisar as diferentes formas de transmissão de dados no universo de Internet das Coisas, assim como seus padrões de comunicação e protocolos. Visa, ainda, estabelecer uma comparação entre dois dos mais usados protocolos de comunicação na camada de aplicação.

Neste sentido, o artigo está organizado da seguinte forma: A seção 2 aborda, de maneira introdutória, os conceitos de Internet das Coisas e suas camadas de protocolos de comunicação, cobrindo alguns dos protocolos de rede e transporte utilizado em Internet das Coisas. Protocolos de aplicação como o CoAp e MQTT são apresentados na seção 3. Por fim, na seção 4 são apresentadas as conclusões e uma indicação do desenvolvimento de estudo de pesquisa.

A.2 INTERNET DAS COISAS

Internet das Coisas, também conhecido como IoT, é um paradigma tecnológico, que tem por finalidade conectar aparelhos eletrônicos utilizados em nosso cotidiano através da internet. A ideia básica é a presença pervasiva ao nosso entorno de uma variedade de objetos, como sensores, atuadores, smartphones, os quais são capazes de interagir uns com os outros e cooperar com objetos vizinhos a fim de atingir um objetivo em comum (ATZORI, 2010). Ou seja, um mundo onde a internet estará em todos os lugares e objetos.

A IoT tem o potencial de mudar a maneira como vivemos. Influenciando tanto o campo doméstico quanto no ambiente de trabalho. No conceito doméstico, a IoT estará aplicada nas casas automatizadas, assisted living, e-health e enhanced learning entre outros diversos usos (ATZORI, 2010), já no ambiente de trabalho, os benefícios afetarão a área da automação, manufatura industrial, logística e transporte de pessoas e encomendas.

A.2.1 Protocolos IoT

Iremos utilizar como modelo de referência, o modelo TCP/IP, usado para os protocolos na Internet. Pelo fato da Internet das Coisas ter sido baseado a princípio na internet, o modelo TCP/IP consegue descrever com naturalidade as camadas utilizadas na mesma. O modelo é dividido em 4 camadas, Rede, Internet, Transporte e Aplicação (TANENBAUM, 2010).

Camada de Rede A Camada de rede é definida como física, ou seja, corresponde às tecnologias usadas para troca de pacotes. No caso da IoT, a comunicação na camada de rede se baseia em comunicação sem fio. Diversos padrões foram especificados para esta camada tais como a série IEEE 802.11, 802.15 (SUTARIA; GOVINDACHARI, 2013).

Camada Internet A tarefa da camada inter-redes é entregar pacotes onde eles são necessários (TANENBAUM, 2010). A idéia dos protocolos desta camada é endereçar, entregar e evitar congestionamentos.

Camada Transporte No modelo TCP/IP, a camada localizada acima da inter-redes é chamada "camada de transporte". A finalidade desta é permitir que as entidades partam dos hosts de origem ao de destino, mantendo uma conversação (TANENBAUM, 2010).

Camada Aplicação Protocolos de serviço específico de comunicação de dados em um nível processo-a-processo. O protocolo de aplicação mais utilizado para prover serviços web é o HTTP, porém possui muita complexidade computacional e consumo de energia elevado para os dispositivos IoT.

A.3 ESTUDO SOBRE PROTOCOLOS IOT

Esta seção aborda alguns protocolos que serão utilizados no estudo, devido sua importância em termos de esforços de entidades reconhecidas na área de redes.

A.3.1 Padrão IEEE 802.15.4

O grupo da IEEE 802.15 específica uma variedade de 'wireless personal area network' (WPANs), entre eles Bluetooth, UWB, BAN, e diversas outras, o padrão estudado neste trabalho - 802.15.4 - representa o maior padrão para low-data-rate WPANs (FRENZEL, 2013). O principal objetivo do 802.15.4 é prover uma base para que outros

protocolos possam ser adicionados nas camadas superiores (transporte, internet e aplicação).

Referido padrão de protocolo atua na camada de rede no modelo TCP/IP, enquanto que as camadas físicas e enlace atuam no modelo OSI. Assim, a camada física é responsável pela frequência, consumo de energia, modulação e a camada de enlace define o formato do dado (FRENZEL, 2013). O IEEE 802.15 é adotado por diversos grupos que atuam no desenvolvimento de objetos e protocolos IoT, como Zigbee Alliance e Thread Group. Este padrão define protocolos e regras de interconexões para comunicação de dados entre dispositivos utilizando baixa taxa de transferência, baixo consumo e baixa complexidade, utilizando rádio frequência de curto alcance.

ZIGBEE

Zigbee é um conjunto de especificações desenvolvido pela Zigbee Alliance para utilização em smart-home e IoT, que define as camadas subsequentes às camadas estabelecidas pelo IEEE 802.15.4, oferecendo serviços de segurança, tolerância a erros e conexão de novos dispositivos. Atualmente, é a tecnologia mais utilizada que faz uso do padrão (FRENZEL, 2013). O Zigbee abrange as camadas referentes a internet, transporte e de aplicação do modelo TCP/IP.

THREAD

Outro conjunto de especificações para utilização em smart-homes e IoTs, assim como o Zigbee, o Thread define camadas a serem utilizadas em cima do padrão IEEE 802.15.4, porém não fornece a camada de aplicação. Utiliza-se sempre de padrões abertos de protocolos para garantir integridade dos pacotes, utilizando os protocolos UDP e 6LoWPAN. Fornece também serviço de segurança e tolerância a erros (THREAD, 2015).

A.3.2 6LoWPAN

Protocolo de permissão para que pacotes IPv6 sejam transmitidos em redes de baixo consumo energético. Seu conceito inicial era permitir que os protocolos de internet se estendessem para todos os tipos de dispositivos (MULLIGAN, 2007). Referido protocolo segue os padrões

definidos no RFC 4944 e RFC 6282, nos quais especificam transmissões IPv6 e sua compressão para camadas de rede IEEE 802.15.4. Sua principal tarefa é comprimir o cabeçalho TCP/IP e também a mensagem enviada, diminuindo o custo de transmissão. Cabeçalhos TCP/IP possuem 128 bytes enquanto IPv6 apenas 40 bytes. Outras tarefas importantes do 6LoWPAN é a fragmentação e reagrupamento de pacotes, assim como a sua distribuição dentro da rede.

A.3.3 Camada de Aplicação

Serão analisado dois protocolos emergentes no mundo da Internet das Coisas. Ambos apresentam características de baixo consumo, por possuírem mensagens pequenas, gerenciador de mensagens e pequeno overhead, ideal para a utilização em dispositivos embarcados (STANSBERRY, 2015). O primeiro protocolo data de 1999, Message Queuing Telemetry Transport (MQTT), apesar da idade, seu uso vem ganhando força com o advento da IoT. Enquanto que o segundo, Constrained Application Protocol (CoAP), data de 2014 e foi desenvolvido especificamente para IoT.

MQTT

Inventado pela IBM para comunicação entre satélites e equipamentos de extração de óleo, possui confiabilidade e baixo consumo como requisitos de sua implementação, o que o torna grande candidato a ser aplicada nas redes IoT. Utiliza o modelo "publish/subscribe" e necessita de um broker MQTT para gerenciar e rotacionar as mensagens dentro da rede. É descrita como um protocolo de comunicação many-to-many. Uma das grandes vantagens do MQTT é a eficiência energética do modelo 'pub/sub', que também escala muito bem. Por utilizar o protocolo TCP, o MQTT já vem munido de segurança na rede. A dependência de um Broker e a utilização do TCP também pode ser um empecilho, pois ambos necessitam de um certo poder computacional para funcionar, impossibilitando o uso MQTT em dispositivos mais simples. Por necessitar de um broaker, este protocolo se torna uma boa opção para comunicação remota/cloud, pelo fato do servidor cloud atuar como o broaker entre o dispositivo IoT e outros serviços.

COAP

Protocolo que utiliza o modelo 'cliente/servidor', o qual disponibiliza interação 'request/response' um-para-um, podendo também suportar multi-cast. Diferente do MQTT, o CoAP surgiu para suprir a necessidade em protocolos IoT, desenvolvido para interoperar com HTTP e com arquiteturas RESTful, através de simples proxies, tornando-se compatível com a Internet. Por utilizar o protocolo UDP, o CoAP apresenta menor consumo computacional e energético. Seu uso permite um menor tempo de resposta quando acionado, pois mantém uma conexão ativa entre nodos. Este protocolo é mais indicado para envio de comandos para nodos locais, por se tratar de uma arquitetura semelhante a HTTP. Também é mais utilizado em dispositivos com menos recursos computacionais, por conta das características explicadas acima.

A.4 CONCLUSÕES

Com o surgimento e desenvolvimento da Internet das Coisas, foi criado, também, o desafio de melhor desenvolver novas formas de comunicação que se adaptam ao conceito, utilizando baixo consumo de energia e baixo poder computacional.

Apesar dos esforços para padronizar e unificar os protocolos de comunicação, com o desenvolvimento de objetos de IoT realizado por diversos fabricantes diferentes, utilizando diferentes protocolos nas camadas mais superiores, como de aplicação, a interoperabilidade se mostrou um dos maiores desafios para essa nova realidade.

Concluímos que, por conta da grande diversidade de protocolos, cada qual com suas características e peculiaridades, se faz necessário um estudo de investigação dos diferentes protocolos disponíveis, para assim, ter uma base de conhecimento, para a escolhas dos mesmos quando for necessário.

A.5 REFERÊNCIAS

Atzori, L. (2010). The internet of things: A survey. Computer Networks.

Frenzel, L. (2013). What's the difference between ieee 802.15.4 and zigbee wireless?

Electronic Design. Mulligan, G. (2007). The 6lowpan architecture.

EmNets '07: Proceedings of the 4th workshop on Embedded networked sensors.
Stansberry, J. (2015). Mqtt and coap: Underlying protocols for the iot. Electronic Design.
Sutaria, R. and Govindachari, R. (2013). Understanding the internet of things. Electronic Design.
Tanenbaum, A. (2010). Computer Networks 5th. Pearson.
Thread (2015). Whitepaper, Thread Stack Fundamentals. Thread Group

ANEXO B – Código Protótipo

B.1 SERVIDOR CENTRAL

Este capítulo do anexo contém todo o código fonte desenvolvido para o Servidor Central e Gateway. O código fonte das bibliotecas utilizadas não foram adicionados, para representar as dependências foi adicionado o código fonte do arquivo package.json, que dispõe de todas as bibliotecas utilizadas, na ordem de dependência, seguida pela versão utilizada no projeto.

B.1.1 /bootstrap.sh

```
#!/usr/bin/env bash

NVM='/vagrant/.nvm/nvm.sh'
NODEJS='/usr/local/bin/node'
LIBSODIUM='/vagrant/libsodium'
PREMAKE='/vagrant/premake-core'
NETCODE='/vagrant/netcode.io'
USER='vagrant'

sudo apt-get update
sudo apt-get -y install build-essential libssl-dev libtool
    git-core curl ntp
sudo apt-get install lua5.2


if [ ! -f ${NVM} ]; then
        cd /vagrant/
        echo "Installing latest nvm"
        su -- git clone https://github.com/creationix/nvm.
            git ~/.nvm && cd ~/.nvm && git checkout `git
            describe --abbrev=0 --tags`
        source ~/.nvm/nvm.sh
        source $HOME/.nvm/nvm.sh
        echo "source ~/.nvm/nvm.sh" >> ~/.bashrc
fi

if [ ! -f ${NODEJS} ]; then
        # install latest stable node.js
        echo "Installing node.js... (please be patient)"
        nvm install stable &> /dev/null
        nvm alias default stable
fi

# install global node packages
echo "Installing global node.js packages... (please be
    patient)"
```

```
# change 'gulp' to 'grunt' depending on project setup
npm install -g npm-check-updates

#Install mongodb
sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80
    --recv 0C49F3730359A14518585931BC711F9BA15703C6
echo "deb [ arch=amd64 ] http://repo.mongodb.org/apt/ubuntu
    trusty/mongodb-org/3.4 multiverse" | sudo tee /etc/apt/
    sources.list.d/mongodb-org-3.4.list
sudo apt-get update
sudo apt-get install -y mongodb-org
sudo service mongod start

mongo
use smartcomm
exit

# install project dependencies and build
cd /vagrant/
echo "Installing local node.js packages... (please be
    patient)"
cd app
npm install
npm start
```

B.1.2 /Vagrantfile

```
Vagrant.configure(2) do |config|
  config.vm.box = "ubuntu/trusty64"

  config.vm.box_check_update = true

    config.vm.network "forwarded_port", guest: 3000, host:
        8001
    config.vm.network "forwarded_port", guest: 3001, host:
        8001

  config.vm.provision "fix-no-tty", type: "shell" do |s|
    s.privileged = false
    s.inline = "sudo sed -i '/tty/!s/mesg n/tty -s \\&\\&
        mesg n/' /root/.profile"
  end
  config.vm.provision :shell, path: "bootstrap.sh"

end
```

B.1.3 /server/package.json

```
{
```

```
  "name": "server",
  "version": "1.0.0",
  "description": "smartcomm",
  "main": "index.js",
  "scripts": {
    "start": "node index.js"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "bcrypt-nodejs": "0.0.3",
    "consign": "^0.1.6",
    "install": "^0.10.1",
    "json-web-token": "^2.1.3",
    "koa": "^2.3.0",
    "koa-bodyparser": "^4.2.0",
    "koa-jwt": "^3.2.2",
    "koa-passport": "3.0.0",
    "koa-router": "^7.2.1",
    "koa-session": "^5.5.0",
    "koa-static": "^4.0.1",
    "mongoose": "^4.12.0",
    "node-libnmap": "^0.2.15",
    "node-nmap": "^4.0.0",
    "passport-local": "^1.0",
    "scan-neighbors": "0.0.3"
  }
}
```

B.1.4 /server/index.js

```
const Koa = require('koa');
const serve = require('koa-static');
const bodyParser = require('koa-bodyparser')
const consign = require('consign');

const mongooseDB = require('./src/bd/mongoose');
const mongoose = require('mongoose');
mongooseDB.connectToMongo(mongoose);

/** Instantiate server application */
const app = new Koa();
app.dirname = __dirname;

/** Set the correct assets and file location used in the app
    */
app.use(serve(__dirname + '/node_modules/bootstrap/dist'))
app.use(serve(__dirname + '/public'))

// body parsing
```

```
app.use(bodyParser())

app.secret = "secret";
const koaJwt = require('koa-jwt');
const jwt = require('jsonwebtoken');

// app.use(koaJwt({secret: app.secret}));

consign({
    verbose: false,
    cwd: 'src'
  })
  .include('models')
  .then('controllers')
  .then('routes')
  .into(app);

app.nodeList = [];
const nmapControl = app.controllers.nmapControl;
nmapControl.retrieveDevices();

app.listen(3000);
```

B.1.5 /server/src/bd/mongoose.js

```
exports.connectToMongo = function(mongoose) {
  mongoose.Promise = require('bluebird')

  var dbConfig = {
    "db": "smartcomm",
    "host": "localhost",
    "user": "",
    "pw": "",
    "port": 27017
  };

  var dbPort = (dbConfig.port.length > 0) ? ":" + dbConfig.
      port : '';
  var login = (dbConfig.user.length > 0) ? dbConfig.user + "
      :" + dbConfig.pw + "": '';var uristring = "mongodb://"+ login
      + dbConfig.host + dbPort + "/"+ dbConfig.db;var mongoOptions =
      db:    safe:  true , useMongoClient:  true ;// Connect to
      Databasemongoose.connect(uristring, useMongoClient:  true );
```

B.1.6 /server/src/controllers/authControl.js

```
module.exports = app => {
  const User = app.models.userModel;
  const jwt = require('jsonwebtoken');
```

```
  this.authUser = async (ctx, next, username, password) => {
    await User.findOne({ username: username }, async
        function (err, user) {
      if (err) {
        ctx.status = 401
        ctx.body = err;
      }
      if (!user || user == null) {
        ctx.status = 401
        ctx.body = {user: false};
      }

      await user.verifyPassword(password, async function(err
          , user) {
        if (!user) {
          ctx.status = 401
          ctx.body = {auth: false}
        } else {
          var token = jwt.sign(user, app.secret);
          ctx.status = 200;
          ctx.body = {auth: true, ip: app.nodeList[0], token
              : token};
        }

      }.bind(ctx).bind(next));

    }.bind(ctx).bind(next));

  }

  return this;
};
```

B.1.7 /server/src/controllers/nmapControl.js

```
var scanner = require("scan-neighbors")
module.exports = app => {

        this.retrieveDevices = async (ctx, next) => {
                await scanner.scanNodes(4000, async function
                    (err, nodes) {
                        app.nodeList = nodes;
                        console.log(app.nodeList)
                });
        }

  return this;
};
```

B.1.8 /server/src/controllers/userControl.js

```
var scanner = require("scan-neighbors")
module.exports = app => {

        this.retrieveDevices = async (ctx, next) => {
                await scanner.scanNodes(4000, async function
                    (err, nodes) {
                        app.nodeList = nodes;
                        console.log(app.nodeList)
                });
        }

  return this;
};
```

B.1.9 /server/src/models/userModel.js

```
const bcrypt = require('bcrypt-nodejs');
var mongoose = require('mongoose');
var Schema = mongoose.Schema;

module.exports = () => {
        const Users = new mongoose.Schema({
          email: { type: String, default: '' },
          firstname: { type: String },
          lastname: { type: String },
          password: { type: String, match: /\w+/, index:
              true },
          username: { type: String, index:  {unique: true,
              dropDups: true} },
        });

        // Execute before each user.save() call
        Users.pre('save', function(callback) {
          var user = this;

          // Break out if the password hasn't changed
          if (!user.isModified('password')) return callback
              ();

          // Password changed so we need to hash it
          bcrypt.genSalt(5, function(err, salt) {
            if (err) return callback(err);

            bcrypt.hash(user.password, salt, null, function(
                err, hash) {
              if (err) return callback(err);
              user.password = hash;
              callback();
```

```
            });
          });
        });

        Users.methods.verifyPassword = function(password, cb
            ) {
          bcrypt.compare(password, this.password, function(
              err, isMatch) {
            if (err) return cb(err);
            cb(null, isMatch);
          });
        };

        return mongoose.model('Users', Users);
}
```

B.1.10 /server/src/routes/authRoutes.js

```
var Router = require('koa-router');

module.exports = app => {
  const authControl = app.controllers.authControl;
  const router = new Router();

  router
  .post('/api/auth', async (ctx, next) => {
    try {
      var auth = await authControl.authUser(ctx, next, ctx.
          request.body.username, ctx.request.body.password);
    } catch(err) {
      console.log(err)
    }
  })
  .post('/api', async () => {
    koaJwt({secret: secret});
  })

  app.use(router.routes());
  app.use(router.allowedMethods());
};
```

B.1.11 /server/src/routes/userRoutes.js

```
var Router = require('koa-router');

module.exports = app => {
  const userControl = app.controllers.userControl;
  const authControl = app.controllers.authControl;
  const router = new Router();
```

```
  /** Server route configuration */
  router
  .post('/api/user', async (ctx) => {
    try {
      userControl.createUser(ctx, ctx.request.body);
    } catch(err) {
      console.log(err)
    }
  })
  .get('/api/user', async (ctx) => {
    try {
      await userControl.getUsers(ctx);
    } catch(err) {
      console.log(err)
    }
  })
  .get('/api/user/:name', async (ctx) => {
    try {
      userControl.getUser(ctx, ctx.params.name)
    } catch(err) {
      console.log(err)
    }
  })

  app.use(router.routes());
  app.use(router.allowedMethods());
};
```

B.2 GATEWAY

B.2.1 /package.json

```
{
  "name": "smartcomm_gateway",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "start": "node index.js"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "body-parser": "^1.18.2",
    "chokidar": "^1.7.0",
    "consign": "^0.1.6",
    "cylon": "^1.3.0",
    "cylon-firmata": "^0.24.0",
    "cylon-gpio": "^0.30.1",
    "cylon-i2c": "^0.26.1",
```

```
    "cylon-raspi": "^0.20.1",
    "express": "^4.x.x",
    "express-jwt": "^5.3.0",
    "jsonwebtoken": "^8.1.0",
    "microcule": "^6.0.0"
  }
}
```

B.2.2 /index.js

```
const express = require('express');
const jwt = require('jsonwebtoken');
const bodyParser = require('body-parser')
const microcule = require('microcule');

/** Instantiate server application */
const app = express();

// body parsing
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({     // to support URL-
    encoded bodies
  extended: true
}));

var auth = microcule.plugins.spawn({
        code: require('./auth-service.js'),
        language: "javascript"
});

const servicesFolder = './services/';
const fs = require('fs');
var port = 4000;
app.services = {};
app.servers = {};
app.listen(port++, function () {
  console.log('server started on port 4000');
});

app.get('/', function (req, res) {
  res.send('SmartComm');
});

app.post('/services', function (req, res) {
        var token = (req.body && req.body.oauth_token) || (
            req.query && req.query.oauth_token) || req.
            headers['x-access-token'];

        jwt.verify(token, 'secret', function(err, decoded) {
      if (err) {
        res.send('Access denied');
```

```
      }

      res.send(app.services);
    });
});

var chokidar = require('chokidar')
var watcher = chokidar.watch('./services')
watcher.on('ready', function() {
  watcher.on('add', function(path) {
    var file = path.split("/").pop();
    runNewService(file, ++port)
  })
  watcher.on('unlink', function(path) {
    var file = path.split("/").pop();
    endService(file)
  })
})

var runNewService = function(file, port) {
        var fileInfo = require(`./services/${file}`)();
        app.services[file] = {};
        app.services[file]['port'] = port;
        if (fileInfo) {
                if (fileInfo.description) app.services[file
                    ]['description'] = fileInfo.description;
                if (fileInfo.params) app.services[file]['
                    params'] = fileInfo.params;
        }
        var service = app.servers[port] = express();

        var handler = microcule.plugins.spawn({
                code: require(`./services/${file}`),
                language: "javascript"
        });

        service.use([auth, handler], function (req, res) {
          // It's good to have a catch-all handler at the
               end in case none the services ended the
               request
          res.end('caught end')
        });

        service.listen(port)
}

var endService = function(file) {
        app.servers[app.services[file]] = null;
        delete app.servers[app.services[file]]
        app.services[file] = null;
        delete app.services[file];
}
```

```
fs.readdirSync(servicesFolder).forEach(file => {
        runNewService(file, ++port)
})
```

B.2.3 /auth-service.js

```
module.exports = function(req, res, next) {
  try {
    var qs = require('querystring');
    var jwt = require('jsonwebtoken');

    if (req.method == 'POST') {

      var body = '';
      req.on('data', function (data) {
          body += data;
          // Too much POST data, kill the connection!
          // 1e6 === 1 * Math.pow(10, 6) === 1 * 1000000 ~~~
               1MB
          if (body.length > 1e6)
          req.connection.destroy();
      });

      req.on('end', function () {
        var post = qs.parse(body);

        jwt.verify(post['oauth_token'], 'secret', function(
            err, decoded) {
          if (err) {
            res.setHeader('WWW-Authenticate', 'Basic realm="
                examples"')
            res.writeHead(401);
            res.end('Access denied');
          }

          next();
        });
      });
    }
  } catch(err) {
    console.log('error', err)
  }
}
```

B.2.4 /services/arduinoLed.js

```
module.exports = function arduinoLed(req, res, next) {
```

```
  if (req == null) {
    return {
        description:'Script to toggle led(pin 13) in arduino
            '
      }
  }

  var Cylon = require('cylon');

  Cylon.robot({
    connections: {
      arduino: { adaptor: 'firmata', port: '/dev/ttyACM0' }
    },

    devices: {
      led: { driver: 'led', pin: 13 },
      button: { driver: 'button', pin: 2 }
    },

    work: function(my) {
      my.button.on('push', function() {
        my.led.toggle()
      });
    }
  }).start();
}
```

B.2.5 /services/helloworld.js

```
module.exports = function hello(req, res, next) {

    if (req == null) {
    return {
        description:'desc',
      }
    }

    const qs = require('querystring');

    if (req.method == 'POST') {
      var body = '';
      req.on('end', function () {
        var post = qs.parse(body);

        res.setHeader('WWW-Authenticate', 'Basic realm="
            examples"')
        res.writeHead(200);
        res.end('Hello World!');

      });
    }
```

```
    next();
};
```

B.2.6 /services/raspLed.js

```
module.exports = function raspiLed(req, res, next) {

        if (req == null) {
                return {
                description:'Script to toggle led(pin 11) in
                     raspberry every 1sec',
        }
        }

        var Cylon = require("cylon");

        Cylon.robot({
          connections: {
            raspi: { adaptor: 'raspi' }
          },

          devices: {
            led: { driver: 'led', pin: 11 }
          },

          work: function(my) {
            every((1).second(), my.led.toggle);
          }
        }).start();
}
```

B.2.7 /service/raspPin.js

```
module.exports = function raspiPin(req, res, next) {

  if (req == null) {
    return {
        description:'Script to toggle a choosen pin every 1
            sec in arduino',
        params:'pin:number'
      }
  }

  if (req.method == 'POST') {

    var qs = require('querystring');
    var post = qs.parse(body);
    var pin = post['pin'];
    if (!pin) {
```

```
      res.setHeader('WWW-Authenticate', 'Basic realm="
          examples"')
      res.writeHead(400);
      res.end('Param Pin is required!');
    }

    var Cylon = require('cylon');

    Cylon.robot({
      connections: {
        arduino: { adaptor: 'firmata', port: '/dev/ttyACM0'
            }
      },

      devices: {
        pin: { driver: 'direct-pin', pin: pin }
      },

      work: function(my) {
        var value = 0;
        every((1).second(), function() {
          my.pin.digitalWrite(value);
          value = (value == 0) ? 1 : 0;
        });
      }
    }).start();
  }
};
```

ANEXO C – Configuração GPIO no Gateway

Este capítulo descreve as instruções necessárias para poder acessar os PINs do GPIO do dispositivo Raspberry pi utilizando a biblioteca Cylon.js, a fim de conseguir manipular os sinais enviados e recebidos do dispositivo através dos micro serviços do gateway. Os PINs são os responsáveis pelo controle de atuadores, transceiver e também receber informações sobre temperatura e outros sensores disponíveis para a plataforma.

Para realizar a conexão da biblioteca Cylon.js com o GPIO do raspberry PI é necessário realizar alguns passos.

Este capítulo leva como premissa que o sistema operacional utilizado no Raspberry Pi seja Raspbian. É importante ressaltar que estas instruções funcionam apenas com Raspberry Pi.

Para acessar os pins GPIO, é preciso adicionar o usuário ao grupo da gpio. In order to access the GPIO pins without using sudo you will need to both app the pi user to the gpio group:

C.1 CONECTAND GPIO DO RASPBERRY PI

```
sudo usermod -G gpio pi
```

Também será necessário adicionar a seguinte regra "udev"ao arquivo /etc/udev/rules.d/91-gpio.rules.

```
SUBSYSTEM=="gpio", KERNEL=="gpiochip*", ACTION=="add",
    PROGRAM="/bin/sh -c 'chown root:gpio /sys/class/gpio/
    export /sys/class/gpio/unexport ; chmod 220 /sys/class/
    gpio/export /sys/class/gpio/unexport'"
SUBSYSTEM=="gpio", KERNEL=="gpio*", ACTION=="add", PROGRAM="
    /bin/sh -c 'chown root:gpio /sys%p/active_low /sys%p/
    direction /sys%p/edge /sys%p/value ; chmod 660 /sys%p/
    active_low /sys%p/direction /sys%p/edge /sys%p/value'"
```

C.2 HABILITANDO RASPBERRY PI I2C NO RASPBIAN

É necessário adicionar essas duas entradas no diretório /etc/modules:

```
i2c-bcm2708
i2c-dev
```

As suas entradas mencionadas abaixas devem estar comentadas no arquivo /etc/modprobe.d/raspi-blacklist.conf:

```
#blacklist spi-bcm2708
#blacklist i2c-bcm2708
```

O arquivo /boot/config.txt deverá ser atualizado adicionando o seguinte texto:

```
dtparam=i2c1=on
dtparam=i2c_arm=on
```

Por fim, é necessário que o usuário tenha permissão para acessar a interface i2c, para isso execute o comando abaixo:

```
sudo usermod -G i2c pi
```

Reinicie o Raspberry Pi.

C.3 HABILITANDO PWM OUTPUT NOS GPIO PINS

Para habilitar o PWM output no raspberry é necessário ter o pi-blaster sendo executado em um raspberry-pi é possivel realizar sua intalação a partir do seguinte endereço web:

```
https://github.com/sarfata/pi-blaster
```